HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Information and Natural Sciences
Department of Computer Science and Engineering

Amr Ergawy

Privacy Aware Key Establishment for Publish/Subscribe Infrastructures in Ubiquitous Environments

Master's Thesis

Espoo, June 30, 2008

Supervisors: Ma Prof. Sasu Tarkoma, Helsinki University of Technology
Associate Prof. Peter Sjödin, The Royal Institute of Technology
Instructor: Ma Prof. Sasu Tarkoma, Helsinki University of Technology

.HELSINKI UNIVERSITY OF TECHNOLOGY Faculty of Information and Natural Sciences Degree Programme of Security and Mobile Computing	ABSTRACT OF MASTER'S THESIS

Author	Date
Ergawy , Amr Ibrahim Ahmed	30/06/2008
	Pages
	128

Title of thesis
Privacy Aware Key Establishment for Publish/Subscribe Infrastructures in Ubiquitous Environments

Professorship	Professorship Code
Data Communications Software	T-110

Supervisors	
Ma Prof. Sasu Tarkoma,	Helsinki University of Technology
Associate Prof. Peter Sjödin,	The Royal Institute of Technology
Instructor	
Ma Prof. Sasu Tarkoma,	Helsinki University of Technology

Ubiquitous Computing, UbiComp, is a vision of embedding computing to every aspect of our day to day life. In ubiquitous computing, interaction among communicating entities exists in highly dynamic, large scale and failure prone environments. Publish/Subscribe interaction paradigm, Pub/Sub, can be used to decouple interacting entities in ubiquitous environments by delivering events based on users interests.

In this scenario, securing events dissemination and protecting users' privacy are essential requirements for ubiquitous applications. In this thesis, we propose an encryption key establishment scheme for encrypting disseminated events in ubiquitous Pub/Sub infrastructures. The proposed key establishment scheme considers the dynamism, scalability and failure tolerance issues of ubiquitous environments. More importantly, the generated encryption keys reflect multi level access control polices, which is important to enforce users' privacy polices.

Keywords
Middleware, Publish/Subscribe, Ubiquitous Environment, Privacy Policy, Encryption Key Establishment, Secure Group Communication

To the soul of my father, to my mother, to my wife, and to my lovely daughter

Acknowledgement

Firstly, I would like to thank the European Union for arranging the study Programme NordSecMob, and funding my studies towards the master degree. Also, I would like to thank the NordSecMob management at TKK, in Finland, for their great work coordinating all the issues inside and outside Finland. In addition, I want to thank my teachers at TKK, in Finland, and KTH, in Sweden, for the knowledge and experience they gave me during my studies.

I thank my supervisor, PhD. Associate Prof Peter Sjödin, at KTH in Sweden, for his important comments which guided me to enhance the thesis quality. I specially thank my supervisor and instructor Ma Prof Sasu Tarkoma, at TKK in Finland. His significant, deep, and main scientific suggestions really strengthened the contribution of the thesis. I am proud to mention that Prof. Tarkoma's proposal of poset derived forest, which is a main contribution to the field of content based Publish/Subscribe systems, was the starting point of my final solution to the problem of this thesis.

Last but not least, I thank my family in Egypt for their continuous support. Also, I specially thank my wife for supporting me all the time in Sweden and Finland during my work on the thesis. Finally I thank my lovely new born, Reem for her smiles which gave me a lot of power to finish the work of this thesis.

Table of Contents

List of Figures

List of Design Concepts

List of Procedures

Chapter 1

Introduction

1.1 Overview

Ubiquitous computing is a wide and significant vision to embed computing in our day to day life [1]. Ubiquitous application scenarios mainly depend on embedded systems, sensor nodes, and mobile nodes. This dependency results into two main features of ubiquitous environments, namely dynamism and failure prone [9]. In particular, communicating entities frequently enter and leave ubiquitous environments.

Consequently, interaction among communicating entities in such environments should employ a suitable paradigm which complies with the features of dynamism and failure prone. Fully decoupling among communicating entities is considered as a suitable solution [9]. Publish/Subscribe, Pub/Sub, interaction paradigm, which is based on receiving subscriptions from users and asynchronously publishing matching events, is a pioneer proposal in providing fully decoupling among communicating entities [3]. This makes Pub/Sub a very suitable choice for ubiquitous infrastructures.

Ubiquitous applications require the support of security to guard users' information. Also, they require the support of privacy to provide access control to users' information. In turn, Pub/Sub based interaction in ubiquitous environments must be secured and supported by privacy aware mechanisms. A main aspect of supporting secured Pub/Sub interaction is the generation of encryption keys. This key generation process must be designed with special considerations to the above mentioned features of ubiquitous environments.

In addition, as supporting security is a required underpinning for enforcing privacy, generated encryption keys should reflect the access control levels which are specified by privacy polices. Unifying security and privacy mechanisms is a trend in the recent Pub/Sub research for ubiquitous environments [4]. And it is the top level goal of this thesis. Next we state our problem definition.

1.2 Problem Statement

Securing event dissemination in Pub/Sub infrastructures for ubiquitous infrastructures must consider the dynamism and failure prone features in ubiquitous environments. These two features are aggregated with other feature to describe the *volatile* ubiquitous environment [9]. In this context, a very significant issue is the used encryption key generation mechanism.

The main investigated problem by this thesis is the possibility of developing an encryption key generation mechanism, for secure Pub/Sub based interaction in ubiquitous environments, which is independent from the volatility of these environments. In addition, this key generation process must be privacy aware one, so that generated keys reflect the access levels specified by users' privacy rules.

Pub/Sub uses data structures for storing and maintaining relations among users' subscriptions, i.e., subscription lattices. Inspired by a very recent and significant proposal of secure Pub/Sub event dissemination [4], we assume that subscription lattices of content based Pub/Sub is a suitable candidates as core data structures for designing a privacy aware key generation mechanism to secure Pub/Sub in ubiquitous environments.

We define the problem of this thesis in points as follows:

a. Can we utilize the subscription lattices in content based Pub/Sub to generate privacy aware encryption keys for Publish/Subscribe users?

b. Is the proposed key generation mechanism compatible with the volatility of ubiquitous environments?

c. Is the proposed key generation mechanism able to generate encryption keys which represent the specified access levels by privacy policies in ubiquitous environments?

1.3 Methodology

In order to address the defined problem of this thesis, we follow two main steps. A relevant methodology is followed at each step.

The first step is to design our proposed key establishment scheme. In this step, we follow an incremental methodology. In particular, we start a design attempt with a set of assumptions, which considers the requirements of key establishment schemes for Publish/Subscribe infrastructures in ubiquitous environments; we extract these requirements during the literature review. The assumptions are about the used core data structure for creating the privacy aware encryption keys as well as the entity which acts as a key manager. We start with the simplest assumptions for the first design attempt, and then we may increment their complexity for the subsequent design attempts.

In turn, we design a privacy aware key establishment scheme based on the current assumptions. Finally, we analyze the resulting scheme to evaluate how it meets our defined goals in the problem definition of the thesis. Based on the analysis we decide whether we have to move to another design attempt or not, and how we should start this next design attempt, if any. Consequently, in case of starting a new design attempt, we define a new set of assumptions for it. This process repeats until we reach the most suitable solution to our goals in the problem definition of the thesis.

From the first step, we will have a set of significant candidate solutions. These solutions proceed to the second step, i.e. prototyping. Because, the work of this thesis is classified as technology led research in ubiquitous computing, i.e. as opposite to application lead research [5], developing a proof of concept is an essential step.

Our methodology in this step is to start the implementation from an already existing implementation of the core data structures which we use to build the candidate key establishment scheme around. At the same time, we implement

our design at the lowest level operations so that we do not affect the original functionalities of the used data structures.

Finally, we visualize the results of the prototyped key establishment schemes. We use results visualization as a helper validation and debugging tool. In addition, for solutions with high operational complexity, we define pass and fail test procedures to validate the correctness of the prototype with respect to specific correctness properties.

1.4 Contributions of the Thesis

The contributions of this thesis can be defined in three points. Firstly, this thesis defines the view of users' privacy dimensions [12], e.g. location and time, as attributes in Pub/Sub subscriptions lattices. This view is significant as a basis for privacy aware key establishment based on these subscription lattices.

The second contribution of this thesis is the proposal of poset based key establishment. Poset is a subscription lattice which maintains complete coverage relations among subscriptions [8, 7]. The significance of this contribution comes from this completeness of poset. In particular, we define an extension to this proposal which can be a very significant routing mechanism for encrypted events, using the poset based generated keys, without affecting the working principle of content based routing, which requires the disclosing of the content of these events.

Finally, we come to the third contribution, namely, forest based key establishment. Forest is a subscription lattice which is a poset derived data structure but with more efficient operations that those of poset [37, 38]. Briefly, forest based key establishment is compatible with the volatility in ubiquitous environments, which is achieved by minimal rate of using of key establishment and updating procedures. In addition, forest based key establishment is a clustered and localized mechanism, which makes it compatible to error prone ubiquitous environment.

1.5 Structure of the Thesis

In the next chapter, chapter 2, we start by reviewing the Pub/Sub as an interaction middleware. Then, we review the ubiquitous environment, its features and its requirements for interaction middleware. Finally, we review Pub/Sub proposals to address two main requirements of ubiquitous applications, namely, supporting privacy and providing security.

Afterwards, in chapter 3, we view the key establishment problem of this thesis as a secure group communications problem for large and dynamic groups [20], i.e., Pub/Sub subscribers groups in the ubiquitous environments. We review relevant proposals of group key management, including key establishment and efficient key distribution mechanisms.

Then, in the rest of chapter 3, we review how security levels, which are ensured by inter-group key management schemes, can be mapped to privacy rules, which is a main design guide for our proposal. In addition, we review other design guides which should be considered for group key management schemes. Also, we specify which ones we consider and how we plan to consider them, and which we do not consider at all.

In chapter 4, we start by the categorization of the privacy rules, this important for specifying the features of privacy rules which we consider for our first data structure assumption. Then, we move to our first design attempt, which is based on the trivial assumption of key establishment at the same entity which maintains privacy rules. The first design attempt is analyzed and the results are used to proceed to the second one, i.e., poset based key establishment. After analyzing the result of the second design attempt, we move to the third and the last design attempt, i.e., forest based key establishment. Finally, the analysis of our last design attempt shows its suitability as a solution for the defined problem in this thesis.

Chapter 2

Background

In this chapter, we define the context of the research in this thesis. Firstly, we define both the technological area, i.e. middleware and publish subscribe software infrastructures, and the application area, i.e. ubiquitous computing. Then, we connect both of them by reviewing the requirements of ubiquitous middleware and viewing them from the perspective of publish subscribe middleware.

2.1 Middleware Infrastructures

Middleware provides abstractions to hide the heterogeneity of distributed systems, which makes the distribution as transparent as possible [6]. This includes providing advanced coordination models as well as managing dependability and performance. Middleware infrastructures can be classified based on the coordination model they provide [6]. The first middleware class is the transactional middleware, which defines contracts to ensure consistent state transition of the distributed systems. A second class is the tuple-space based middleware, which makes use of a globally shared storage space that can be accessed by interacting entities for inserting, reading and removing tuples.

Another middleware class is the remote procedure calling middleware. It allows interacting entities to invoke procedures remotely as if they are invoked locally, which makes it a significant interaction model. A natural evolution of the remote procedure calling middleware class is the object and component oriented middleware. Basically, it is a mapping of the object oriented and component based paradigm to a distributed environment.

A fifth middleware class is the message-oriented middleware. It is a variation of the tuple-space based middleware. The difference is that tuples are implemented as messages and the globally shared space is implemented by distributed message-queues. An interacting node registers as a member in a message queue to receive messages. Another alternative to implement a shared space is to deliver messages by means of predicates, which is the work principle of publish subscribe middleware, Pub/Sub. In contrast to all the previously mentioned

classes of middleware, Pub/Sub provides fully decoupled interaction among interacting nodes [3]. This is more explained in the next section.

2.2 Publish/Subscribe Middleware

Publish Subscribe, Pub/Sub, is an interaction paradigm which has received a significant amount of interest from both research and industry [3]. Mainly, Pub/Sub gains its significance from providing loosely coupled interaction among communicating entities in large scale communication environments. As a result, Pub/Sub is a very suitable interaction paradigm for asynchronous, dynamic and failure prone communication environments such as mobile and ubiquitous infrastructures.

2.2.1 Interaction in Publish/Subscribe

Basically, a Pub/Sub interaction scenario includes *publishers, subscribers, event service, events, and notifications* [3]. *Publishers* are entities which produce information to reflect *events* happening at their side. *Subscribers* are entities which are interested in such events. Subscribers submit subscriptions to *the event service* to express their interests. It is the responsibility of the event service to deliver *notifications* about the subscribed events from the publishers' side to the subscribers' side.

The event service is composed of a network of *event routers/brokers,* which are connected by means of communication network. Pub/Sub brokers are mainly discriminated into two categories, viz. *terminal/local brokers* and *intermediate brokers* [14]. Terminal/Local brokers are those brokers at the borders of the Pub/Sub brokers network/cloud. A subscriber/Publisher can access the event service via the Pub/Sub brokers which are local to it. On the other hand, intermediate brokers are more inner brokers in the Pub/Sub network/cloud and their main function is forwarding notifications. In figure.1, a Pub/Sub scenario is illustrated.

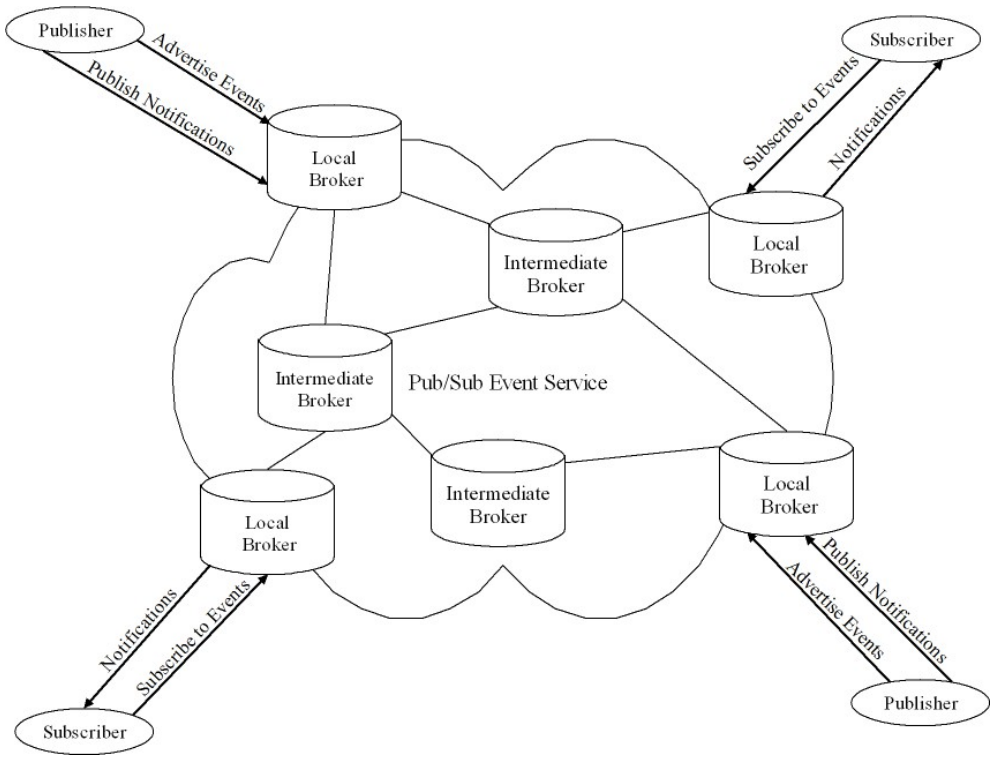

Fig.1. an example of Publish/Subscribe Interaction Scenario

2.2.2 Interaction Decoupling in Publish/Subscribe

Pub/Sub decouples publishers and subscribers at three dimensions, namely, time, space, and synchronization [3]. Firstly, at the time dimension, interacting entities do not have to be connected to event service at the same time to communicate. Secondly, for Space de-coupling, the publishers and the subscribers do not have references to each other. Also, they do not know how many entities of each side are involved in the interaction. Finally, from a synchronization point of view, publishing events or being notified about them doesn't exist in the main flow of control of an interacting entity. The event notification service is responsible for storage and management of subscriptions and efficient delivery of events, which is essential for interaction decoupling the Pub/Sub.

2.2.3 Variants of Publish/Subscribe

Mainly, Pub/Sub has three main variants namely, topic based Pub/Sub, content-based Pub/Sub, and type-based Pub/Sub [3]. In topic based Pub/Sub, a topic is used as an initialization argument in the delivered event and every topic is viewed as an event channel. Topic based Pub/Sub provides no intelligent interaction, because subscribers get all published events under the subscribed topic without any filtering.

As an intelligent variant, content based Pub/Sub filters events based on their properties, e.g. Meta data associated with the event or the internal attributes of the data structures carrying the event. As a result, subscriptions take suitable forms which enable the filtering process. Usually, subscriptions take the string form to describe filtering constraints, which can be logically combined to have complex subscription patterns. In addition, it is possible to have events correlation or logical combinations of elementary events.

A more structured variant is type based Pub/Sub. To deliver an event, it is enough that it complies with a template object. In addition, type based Pub/Sub is intelligent because matched events are not necessary to be the same as the templates, e.g. attributes should match except for those of null value. In the next section we introduce the Pub/Sub service which we start our proposal from it.

2.2.4 SIENA: *A Content Based Publish/Subscribe Service*

SIENA is a very significant proposed content based Pub/Sub service [8, 7]. The interaction in SIENA is similar to the reviewed Pub/Sub interaction scenario in section 2.2.1. From a high level perspective, *publishers* specify the events they may publish using *advertisements*. *Subscribers* specify events of their interest in terms of *subscriptions*. Publishers generate *notifications* about their events. In turn, SIENA, as the *event service*, delivers notifications to the target subscribers.

SIENA is composed of a set of *event servers* [8, 7]. The number and the topology of the interconnected event servers may vary, depending on the

required configuration of the event service. A publisher or a subscriber may access the service via one event sever, which is referred to as access point. At the end, all of the interacting entities in a Pub/Sub service communicate by a communication network, regardless of the exact physical details of this network.

The function of forwarding subscriptions and notifications is done by the event servers, which can be interconnected in a number of topologies [8, 7]. They can be interconnected in a hierarchical topology, where a parent server receives all subscriptions from its children and it only sends notifications back to them. This is a natural extension to the centralized event service. Of course, the main advantage of this topology is that an entire subset of servers can join the network by just connecting their root servers to any connected server to the hierarchy. However, the main problem is the overloading of higher level servers.

Alternatively, event servers can be connected by an acyclic peer to peer topology [8, 7], so that servers communicate with each other as peers. The server to server connections are non-directed arcs. The whole connection set is maintained as an acyclic graph. In figure.2, an example acyclic peer to peer event severs network is illustrated [8].

Different issues of advantages and disadvantages of this topology were investigated discussed [8], and a cyclic peer to peer variant is proposed. In addition, a hybrid approach of combining the acyclic peer to peer topology with the hierarchal topology is introduced. However, for the purpose of this thesis we assume that the acyclic peer to peer topology is the default configuration of the SIENA events servers' networks.

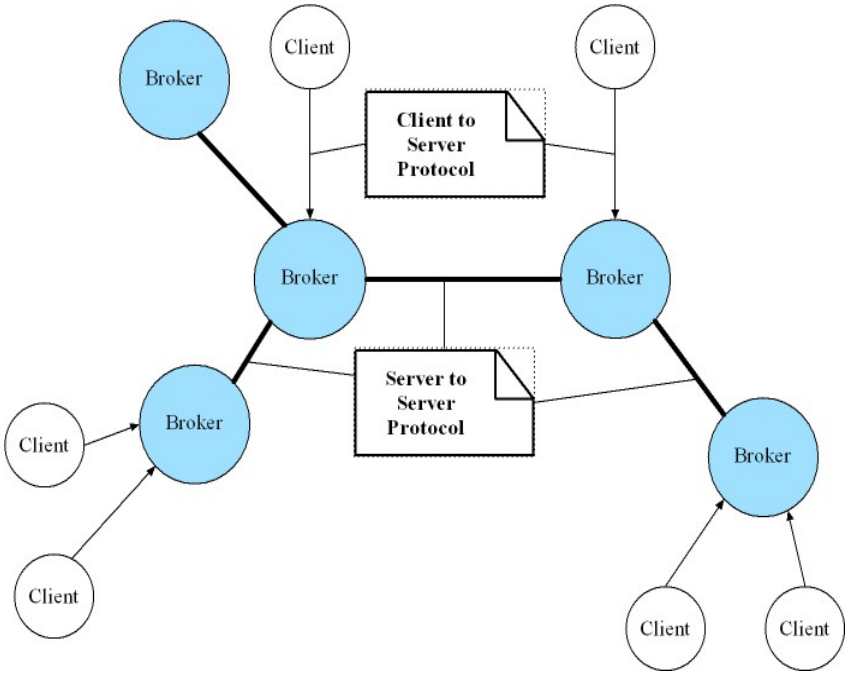

Fig.2. an acyclic peer to peer Publish/Subscribe event servers network [8]

Inside the event service [8, 7], advertisements are used to guide the event service so that it can efficiently route subscriptions towards the publishers, which will produce matching notifications to these subscriptions. On the opposite direction, subscriptions are used to route notifications from their publishers to their subscribers. In other words, both advertisements and subscriptions work together to avoid sending subscriptions to all servers. In particular, SIENA uses advertisements to identify the potential sources of events. Two variant of SIENA can be defined, viz. *subscription based service*, where only subscriptions form the semantics of the service, and *advertisement based service*, where only advertisements form the semantics of the service [8, 7]. For the purpose of our work, we only consider the first variant, namely, the subscription-based service.

In SIENA, an event notification is defined as a set of attributes [8, 7]. And each attribute is a triple of name, type and value. An attribute is uniquely defined by its name. In figure.3, an example of event notification is illustrated [8]. An attribute type is defined from a limited set of data types, which include char, integer, and Boolean. In addition, operators, mainly for matching purposes, are defined by SIENA. These definitions of data types and matching operators are mainly used for matching notifications against subscriptions, which are defined in terms of filters. A filter is defined in terms of a set of attribute names and types by applying constraints to their values. A pattern of interesting events can be defined by combining a set of filters. In fig.4, an example of event filter is illustrated [8].

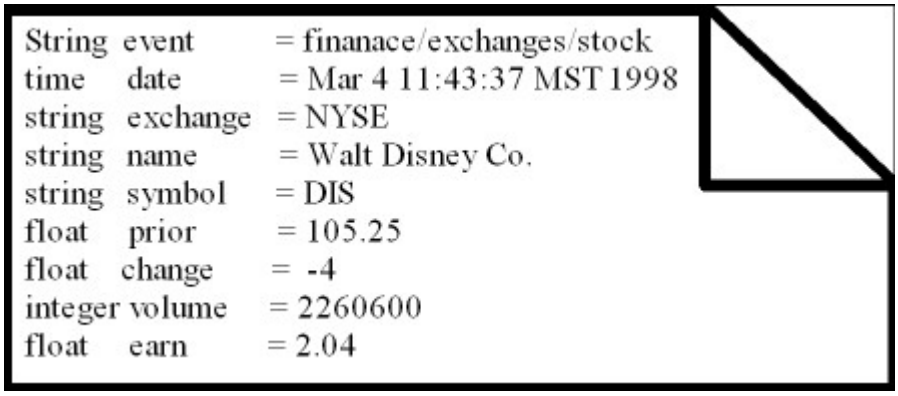

Fig.3. an example of even notification [8]

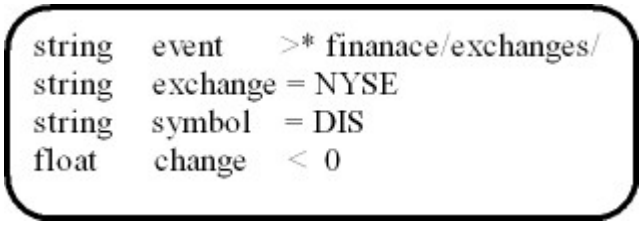

Fig.4. an example of even filter [8]

In this section, we reviewed Publish/Subscriber middleware as the used technology side of the research context of this thesis. Also, we introduced SIENA, the Pub/Sub content based service proposal which we start our proposal from. In the next section, we introduce ubiquitous computing, the application area of the proposal in this thesis. Moreover, we review the interesting characteristics of the ubiquitous environments, where the fully decoupling of interaction in Pub/Sub can be very suitable.

2.3 Ubiquitous Computing

Ubiquitous computing was proposed as a vision to spread computing as an essential part of day to day life [9, 1]. Computing devices are embedded in surrounding physical objects and places. The connection among this variety of appliances is done by means of communication networks, which are typically wireless networks. Controlling such physical objects and appliances may be done by various interaction devices, which may be mobile devices. However, a user may not be concerned with any control process. He even may not be aware of the existence of the surrounding computing environment while it is serving him.

Research in ubiquitous computing focuses on small special purpose devices, special network protocols to communicate these devices, interaction infrastructures, possible ubiquitous applications, wireless communication, location and resource discovery, power consumption, security as well as privacy [2]. A more focused concept is *pervasive computing*, which is concerned with how people see mobile and wireless computing devices, how application are developed and deployed, and how ubiquitous infrastructures serve the pervasive environment [2]. The work of this thesis is focused at the ubiquitous infrastructure side.

Next, we review a set of characteristics of ubiquitous environments which impose its significant challenges on designing software systems for these environments.

2.3.1 Characteristics of Ubiquitous Environments and Software Challenges

In ubiquitous computing, software is required to work in our everyday applications, on failure prone hardware with limited resources, and in continuously dynamic environments [9]. These requirements result from two main characteristics of the ubiquitous systems. The first characteristic is the physical integration among resource constrained embedded and sensor nodes from one side and objects in the real world from another side.

The second characteristic is the spontaneous interoperation, in ubiquitous environments, among units of software which can be services, clients, resources or applications [9]. In general, such software components are usually working on nomadic devices, which continuously come to/leave the environment. Moreover, the availability and features of a software component/unit may itself vary with time. For example, a software component in a spontaneous interaction may communicate with a set of other software components/units which may change their identities and functionalities with time in response to the surrounding environment.

Similarly, this spontaneous interaction in ubiquitous environments applies to users and hardware. At a more general scope, spontaneous interoperation is defined by the *Volatility Principle*, a design principle which we must follow when designing any system for ubiquitous environments.

> ***The Volatility Principle***
> *"You should design ubicomp systems on the assumption that the set of participating users, hardware, and software is highly dynamic and unpredictable. Clear invariants that govern the entire system's execution should exist"* [9]

This principle reflects the high dynamic nature of the ubiquitous environment, which is a main guide line followed by the work of this thesis.

Considering the two above mentioned characteristics of ubiquitous environments and the volatility principle, the design of ubiquitous software infrastructures should consider issues like adapting to dynamic environments, fault tolerance, and resources constrained devices [9]. In the next section, we review the requirements of ubiquitous middleware.

2.3.2 Requirements of Ubiquitous Middleware

At the middleware level of ubiquitous infrastructures, common requirements are imposed by different applications [10, 2]. These requirements include handling mobility of users and communicating nodes, the ability of self organizing and stabilizing to adapt with the dynamic and failure prone environment, the ability to handle device heterogeneity in the ubiquitous environments, and providing security to guard user data against unauthorized access or modification [10, 2]. In addition, these requirements include user intent awareness, context awareness and adaptation, delegation, high-level energy management, and managing privacy and trust [6]. For the purpose of this thesis, we focus on the requirements of providing security as well as privacy.

Privacy and Trust

In order to support different functionalities of ubiquitous applications, intensive management and distribution of user information is required [6]. Trusting the ubiquitous environment is a critical issue which should be considered at all levels, including the middleware level. From the users' perspective, ensuring privacy is an essential requirement in order to avoid undesired use of their data. The work of this thesis considers protecting users' privacy by means of suitable encryption schemes.

Privacy Policies in ubiquitous Environments

In ubiquitous environments, users should be able to specify policies to control the distribution of their information, e.g. location information [11]. Even if there are no assumptions of intended violation of the privacy rules, inadvertent violations are still possible. Software

infrastructures, e.g. Pub/Sub substrate, are responsible for enforcing the specified policies.

An example of privacy policies is the context based information sharing, e.g. with respect to time and location. Another example is transforming the information of user location into anonymous forms for specific use cases, e.g. hiding the name of a user or abstracting its location inside a building [11]. Moreover, richer privacy policies include restricting the ability of users to delegate their rights to other users.

Survivability and Security

From a security perspective, there are two types of system survivability namely, survival by protection, SP, and survival by adaptation, SA [2]. As for SP, access control and encryption are used to protect applications from both accidental and malicious emerging harms. On the other hand, SA attempts to adapt the system to face the changing conditions. In this thesis, we focus on providing SP by means of encryption at the level of ubiquitous middleware.

Security for Resource-Poor Devices

In section 2.3.1, we mentioned that ubiquitous software infrastructures should consider the limited resources of the physically integrated/embedded computing devices. Such limitations have impact on the used security mechanisms [9]. For example, extremely resource poor devices may not be bale to perform public key encryptions. As a result, in such case, only symmetric key encryption is used. Even if the computational resources allow using public key encryption [29], then using it must be minimized because its power consuming computations. This issue is a main guideline for this thesis.

In the next section, we view these requirements of ubiquitous middleware from the perspective of Pub/Sub middleware.

2.4 Publish/Subscribe as a Ubiquitous Middleware

Many of the mentioned requirements of ubiquitous middleware, in section 2.3.2, are concerned with the used interaction paradigm. In this section, we review how these requirements are addressed by Pub/Sub infrastructures. Next, we review how privacy is supported in Pub/Sub infrastructures.

2.4.1 Supporting Privacy in Publish/Subscribe Middleware

In Pub/Sub, subscribers specify what events they want to receive. Oppositely, privacy policies enable publishers to specify who receive their information and under what terms [11]. At this point, two roles must be discriminated viz. anonymous collection of information and privacy policy controlled dissemination of events, which is the wide problem domain of this thesis.

Anonymous Collection of Users' Information

Anonymous collection of users' information requires querying these information considering issues like trust and delegation of rights. A very important example is the anonymous collecting of users' location information. This is achieved by giving a user, who may be a Pub/Sub publisher, the right to choose how the event service can access his information [11]. For example he may specify that his information can be distributed only during specific hours in the day, e.g. work day hours, and even while he is in specific locations.

However, a publisher does not have a default right to access any aggregated information by the service, which already includes his own information. It is the responsibility of the service administrator to delegate rights to interacting entities, including the ownership of event types, adding new event types, publishing events, subscribing to events, and even the ability to delegate access rights to other entities. Based on the delegated rights, interacting entities can perform actions. Enforcing these delegated rights affect the events dissemination in Pub/Sub in ubiquitous environments, which is discussed in section 2.4.2.

A more powerful tool that may be given to a user of an event service is the ability to specify the level at which his own information can be disclosed. This is provided by enforcing the concept of *Information Precision* [12]. This is an essential issue for privacy enforcement in ubiquitous environments. It should not be confused with information accuracy issues regarding the noise of the sensors in ubiquitous environment and their efficiency degradation. More specifically, Information Precision means that users can control their privacy by specifying the precision of disclosing their own information.

As opposite to disclose/non-disclose privacy policies, information precision enables multilevel privacy policies [12]. For example, a user may specify that his location information can be only disclosed in terms of buildings names and not in terms of exact room numbers. At the end, specifying information precision uses precision scales, which may differ based on the types of information they represent. A general example of such scale from highest to lowest can be represented as follows:

{Precise, Approximate, Vague, Undisclosed} [12]

In ubiquitous applications, privacy is commonly concerned with the following set of personal information:

{Identity, Location, Time, Activity, and Nearby people} [12]

An example of applying the above mentioned precision scale to these types of information is illustrated by in table.1 [12].

To sum up, anonymous collection of users' information functions as the front end of supporting privacy by Pub/Sub middleware, we move now to the back end, i.e. privacy policy controlled event dissemination.

Table.1.Normalized precision levels of personal information dimensions [12]

	Identity	**Location** Indoor/outdoor	**Activity**	**Nearby People**	**Time**
Precise	True name	Room/Block	Precise	Names	In Minutes
Approximate	Pseudonym	Building/ District	Categorical	Roles	Hours
Vague	Role	Municipality	Busy/ Not Busy	Number	Days
Undisclosed	Undisclosed	Undisclosed	Undisclosed	Undisclosed	Undisclosed

Privacy Policy Controlled Event Dissemination

As we mentioned in the previous point, entities in a privacy aware Pub/Sub infrastructures interact with each other based on delegated rights. Also, these rights are affected by the ability of the users to control the precision at which their information is disclosed. In particular, a user affects the rights of other users by specifying his privacy policy.

To specify the delegated rights to users, policy condition rules are defined [11]. These rules are logical predicates which include conditions applied to the allowed Pub/Sub action by different interacting entities. For example, a logical predicate may describe whether an entity is allowed to publish/subscribe to a specific event type. Once that entity attempts to perform a Pub/Sub action, e.g. subscribing to an event, the privacy aware infrastructure evaluates the relevant predicates, which results into a Boolean result. Based on that result, the entity may proceed or not in accomplishing the attempted action. An example of this process is shown in figure.5 [11].

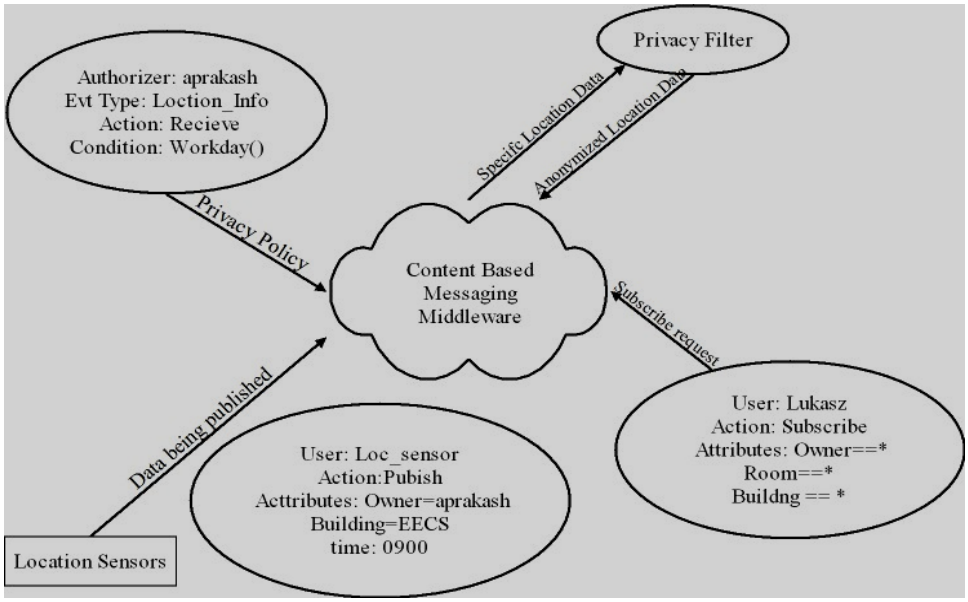

Fig.5. Privacy Controlled Event Dissemination [11]

In addition to applying condition rules to Pub/Sub operations, these rules may apply to any possible event attributes or external attributes [11]. These attributes may be privacy attributes like those we mentioned in the previous point of this section. For example, in a location aware ubiquitous environment, an event attribute can be the location of the user and an external attribute can be the time at which this location event is collected. A logical predicate of a condition rule specified by that user may include applied conditions to these two privacy attributes.

To provide privacy policy controlled dissemination of events, a privacy aware Pub/Sub infrastructure must support two facilities, namely, the definition and evaluation of the privacy condition rules [13] and the provision of security underpinnings [11].

a. Definition and Evaluation of Privacy Rules

The first side of providing this facility is supporting high level of expressiveness to rules building tools, so that they are flexible enough to describe a wide variety of privacy rules. The second side of this facility is supporting efficient evaluation of these privacy policy rules. A trade off between these two sides is required, since the higher the expressiveness the less the rules evaluation efficiency.

A recent proposal which achieves the balance between efficiency and expressiveness is CPOL [13], a framework for policy evaluation. CPOL employs both an efficient evaluation mechanism as well as caching mechanism. Also, it provides the expressiveness levels similar to those provided by database engines like MySql. For the performance, CPOL goes higher by six orders of magnitude than MySql, which has a throughput of only few thousands queries per second. CPOL is designed to serve in large-scale infrastructures, with a size of 10,000 or more users.

Building and Evaluation of policy rules has already received a significant amount of research efforts and reached a good level of performance. This functionality is out of the scope of the work of this thesis.

b. Security Enforced Privacy Policy

As we mentioned above, it is important that security underpinnings support privacy policy controlled event dissemination. This facility is required to enforce privacy in entrusted environments [11]. The situation gets more complicated in dynamic large scale ubiquitous environments, where spanning several administrative domains requires addressing special security considerations [14].

For more analysis, we view the problem of enforcing the privacy policy condition rules as an access control problem. For a Publish/Subscribe

infrastructure in a ubiquitous environment, access control functionalities can be done to specialized entities [14]. This decision can achieve efficiency, by not overloading the brokers of the event service, as well as limiting the trust assumptions [15], which we discuss in section 2.4.3.

At the same time, to provide encryption of disseminated events, as a security underpinning for enforcing privacy policy, specialized entities are required to establish encryption keys, which is the main focus of our work in this thesis. A very reasonable solution is to move both functionalities, namely, access control and key establishment to the same entity in the network. In a security enabled Pub/Sub substrate, these entities may be called key managers [16, 14]. Consequently, such decision can be a first step towards achieving the long term goal of unifying privacy mechanisms and security mechanisms in Pub/Sub infrastructures [11, 4], which is a goal of our work in this thesis too.

In this section we reviewed how Pub/Sub infrastructures can support privacy in ubiquitous environment. Two main sides were reviewed, namely, the anonymous collection of users data and the privacy controlled event dissemination. At the end of this section, we came to the point were security underpinnings are required to enforce users privacy rules.

In our proposal, we focus on encrypting events in privacy aware manner. In turn, we have to review how encrypting events in Pub/Sub may affect the working principle of content based Pub/Sub, i.e., routing events based on their disclosed content. This issue is discussed in the next section.

2.4.2 Event Encryption for Content Based Publish/Subscribe

In Pub/Sub infrastructures, an event is encrypted using encryption keys which are managed by a set of entities, i.e. key managers [16, 14]. A publisher, a subscriber, or an event router/broker can operate on events or events attributes only if it has the appropriate encryption keys. Any of such entities must authenticate itself to the key managers to receive encryption keys. This is an

issue of both trust and authorization, which we discuss in section 2.4.3. In addition to their main task of managing encryption keys, key managers control access to these keys [14].

In such large scale and multi-domain environments, like ubiquitous environments, Pub/Sub brokers in some administrative domain may not be allowed to access the contents of the events from other domains [14]. At the same time, as we reviewed about content based Pub/Sub in sections 2.2.3 and 2.2.4, these brokers have the duty of routing events, which requires disclosing the content of events. As a result, a question mark originates about how an event should be encrypted to enable brokers to make content based routing decisions. Answering this question has received a significant amount of research. In this section, we review two proposals with significant contributions.

Multilevel Event Encryption and Event Type Based Routing

In the first proposal [14], an event type is given a unique global identifier [15], which is used by key managers to generate encryption keys for this particular event type. When an authorized publisher wants to publish an event, it registers itself to an authorized local broker. Then, the publisher submits the event to this broker. In turn, the local broker uses the encryption key, which is associated with the type of the received event, to encrypt the entire event. Afterwards, the local broker forms a message which contains both the encrypted event as well as its unique global event type identifier. Finally, the authorized local broker pushes the event in the network towards subscribing intermediate brokers, which some of them may be authorized to see the contents of the event or not.

Once it receives an encrypted event message, an authorized intermediate/local broker extracts the combined unique global identifier of the event type. The broker uses this identifier to specify which of the encryption keys it has is associated with this event type. It’s important to note that the authorized broker has already got this encryption key of this event type from the key managers. After the authorized broker decrypts the event, it uses the disclosed event

attributes to make a content based routing decision. Finally the broker forwards the received encrypted event message further into the Pub/Sub broker network.

Oppositely, once an unauthorized broker receives an encrypted event message, it will forward it only based on the combined unique global identifier of the event type. In other words, it forwards all the relieved events of the same type to all subscribers/brokers who are interested in that event type, without making any intelligent content based routing decisions. This is a very significant limitation of this event encryption scheme. However, this scheme has the advantage of doing only one encryption operation at the start of the event dissemination. A similar concept to this unique global event type identifier based forwarding is referred to as Token Based Forwarding [4]. It is again an event type representative for routing encrypted events.

In addition, this event encryption proposal defines attribute level encryption [14]. It defines a globally unique attribute identifier for each attribute in a given event type. This identifier is used by the key managers to generate an encryption key specifically for encrypting the values of this attribute. Encryption at the attribute level introduces higher overhead. However, it enables users to define more expressiveness access control on their events at the attributes level.

In attribute based encryption [14], an event message includes the globally unique identifier of the event type. Again this identifier is used for event forwarding at the unauthorized brokers. As a result, again no intelligent content based routing decisions are done at the unauthorized brokers. Again, the only advantage is the given expressiveness to the user to define his access control polices at the attribute level.

In an event message, the globally unique identifiers of encrypted attributes are included. Authorized brokers use these identifiers to know the proper key which they will use to decrypt the attribute values in a received encrypted event message. Again, authorized brokers have already got these keys form the key

managers. Consequently, they can make intelligent content based routing decisions using the decrypted event attributes. In the next section, we review another significant proposal for attribute based event encryption. It defines the possibility of encrypting some attributes of the event and disclosed others, for making content based routing decisions using them.

Selective Attribute Based Event Encryption

In another significant event encryption proposal [16], an assumption of not trusting the broker network at all is made. At the same time, without using the globally unique identifier of event types proposed, of the previously discussed proposal, an encryption of the entire event is excluded [16]. In general, matching and routing based on an entirely encrypted event is very difficult to achieve in a practical way. In particular, suggested techniques to perform computations on the encrypted data can not be practically implemented [16]. Also, this may require modification to existing matching and routing algorithms.

Consequently, disclosing some attributes of an encrypted event to make routing decisions is required. At this point, a very important note comes to the image which states that: *"Only selected parts of the events need to be confidential, matching and routing should be accomplished without these parts"* [16]. For example, in a ubiquitous location-aware service, the attribute of user's identity can be disclosed while the value of his location or the value of the time at which he was in that location may not be disclosed. As a result, routing can be done based on the disclosed attribute, i.e. user identity. This is more intelligent than making routing decisions only using the identifiers of the event types, as discussed in the previous proposal.

In turn, we need to make use of a selective approach to select which attributes of an event to encrypt and which to disclose [16]. In particular, two main issues arise from such selective approach. Firstly, a limitation of the content based routing capability is very likely to exist. This is because the encrypted attributes can not be used for making content based routing decisions. Secondly, a

question mark arises about the forms of confidentiality categories or descriptions which can be used to describe the selection process of attributes for encryption.

In this section, we discussed two significant proposals for event encryption in Pub/Sub middleware. The most important issue for both of them is how event encryption affects the content based routing of events. Basically, two main approaches of event encryption are proposed, i.e. event based encryption and attribute based encryption. In the proposed solution in this thesis, we follow the event based encryption approach. For both approaches, the role of key management is delegated to trusted entities, e.g. Pub/Sub brokers, which is a matter of trusting these entities. In the next section we review trust in Pub/Sub infrastructures.

2.4.3 Trust in Dynamic Publish/Subscribe Infrastructures

From section 2.4.2, we can conclude that two issues of the event encryption process are related to trusting entities in a Pub/Sub infrastructure. The first one is the allocation of the key management functionality. The second one the classification of Pub/Sub brokers into authorized or unauthorized with respect to the ability to access the contents of an event. More importantly, this trust issue gets more complicated when we consider the continuously dynamic Pub/Sub substrate in ubiquitous environments [17].

Dynamism in Ubiquitous Publish/Subscribe Infrastructures

In ubiquitous environments, a Pub/Sub substrate is required to be flexible to provide better applicability as well as to reduce the required management efforts [17]. This flexibility requirement is even strengthen by the need of both collaboration and fast events dissemination in the ubiquitous environments. Moreover, the common assumption of statically or manually managed topology of notification service brokers is very clear to fail for a Pub/Sub substrate functioning in a ubiquitous environment. Additionally, usage pattern in these environments are highly dynamic. Together all of these reasons form the source of dynamism in a Pub/Sub infrastructure in a ubiquitous environment.

For example, in an e-home scenario [17], all networked device may interact thought a Pub/Sub substrate. According to the volatility principle, which is reviewed in section 2.3.1, nodes used by users at home continuously join and leave the network. Additionally, there is no assumption about the existence of a skilled administrator to maintain the system configuration. As a result, the Pub/Sub substrate must be able to adapt itself to different usage patterns and different failures. Consequently, the need of dynamic Pub/Sub substrates in ubiquitous environments is a must, which in turn moves the issue of trust to a higher complexity level.

Back to the trust issue, it is clear that in a static Pub/Sub substrate topology we can depend on building trust domains [14]. On the other hand in topologically dynamic Pub/Sub substrates, we need to enforce access control and confidentiality among all involved entities, including Pub/Sub brokers. This requirement implies the mentioned two issues at the beginning of this section, i.e., the allocation of the key management functionality and the authorization to access events.

Trust and Authorization of Event Access in Publish/Subscribe Infrastructures
Looking at authorization as a result of trust, we review a significant proposal which joins the two concepts, namely, Role Based Access Control, RBAC [18]. In this proposal, roles are defined and can be activated by interacting entities in the Pub/Sub substrate. In RBAC, roles function as a security basis for controlling publications, subscriptions, and other related operations. Role based access control is very suitable to a continuously evolving and dynamically changing Pub/Sub substrate. RBAC defines the concept of event type owner, which are assumed to be trusted by a Public Key Infrastructure, PKI. This is a very small scale trust assumption which RBAC starts with.

Based on the trust in type owners, they are authorized to have control over the policy which applies to accessing their own event types. At the same time, a type owner does not have to be a publisher of this type. Consequently, even if

there are no publishers or subscribers in the Pub/Sub substrate, registered event types along with their event owners always exist. This is quite suitable to a Pub/Sub substrate in a ubiquitous environment. As for an event type, it can be defined in terms of its name and its attributes [18]. Event type definitions are kept in a logical event respiratory, where they are organized in an inheritance hierarchy. The storage enables event types to be checked at publishing time and the hierarchy enables multilevel access control.

In RBAC, roles relate principals and privileges [18]. Moreover, RBAC states that the policy which shapes such roles must be dynamic and loosely coupled from the protected software itself. In an RBAC system, roles are activated by an entity in the Pub/Sub network to make use of their privileges. An entity must be authenticated before being able to access any roles. An entity may activate more than one role. A collection of activated roles is defined as a session, which facilitates the simultaneous management of these roles. RBAC may require the activation of some roles by an entity to be able to proceed with activating further roles.

More specifically, subscribers and publishers can determine what events they are permitted to subscribe to /publish based on the specified access policy by the type owner as well as the credentials provided by them [18]. This assumes that at least the local brokers to these publishers and subscribers are trusted by the type owners. For the credentials which can be used to design access control policy, there are two broad categories, viz. authentication and capabilities. Also, other factors can be included in the credentials, e.g. user's location. At this point we have two important notes; firstly, the scope of trust has been widened to include local brokers in the Pub/Sub event service. Secondly, the trusted event type owner gave access authorization to publishers and subscribers. These two notes are more discussed as follows:

a. Fully Trusted Broker Network

As we mentioned above, RBAC defines event types as well as their owners. This definition targets to make few extra steps for publishers

and subscribers to be able to operate on a registered event type. In turn, access control overhead is lowered as much as possible. Moreover, RBAC targets to keep access control presence as hidden as possible to publishers and subscribers.

From the overhead point of view, a special consideration is given to publishing events, since this process is usually more frequent in Pub/Sub than the process of subscribing events. In turn, RBAC aims to make access control overhead per event publication as low as possible. RBAC achieves this by trusting brokers in the Pub/Sub substrate. This trust assumption enables local brokers to handle the security checks for them to lower access control overhead [18].

b. Authorization of Event Access

As mentioned above, in RBAC different entities in the Pub/Sub substrate must provide the proper credentials to access a specific event type. One important category of credentials is defined in terms of assigned capabilities interacting entities [14]. In particular, event type owners grant authorization capabilities to entities in a Pub/Sub substrate by granting authorization certificate with specific access rights [15]. If this certificate allows an entity to delegate its own rights to other entities, then this authorized entity can grant authorization certificates to other entities. And this process repeats. Finally an event type owner can from a certificate chain to decide whether an entity is allowed to perform some action or not.

Consequently, if an entity is allowed to access a specific event type, then it is allowed to access its encryption keys. Additionally it is important to mention that the capabilities assigned to a publisher/subscriber must be assigned to its local broker in the Pub/Sub substrate [18, 14], because local brokers are the access point of this client to the broker network [16, 15, 14].

To sum up, the assumption of fully trusted brokers can not be always realistic. Additionally, as we mentioned in section 2.4.2, access control must be enforced by confidentiality, which requires encryption. In turn, a higher level of authorization is required to define the previously mentioned key managers. These two issues are reviewed in the next point.

Limited Trust in Publish/Subscribe Infrastructures

In a ubiquitous environment with a large and dynamic Pub/Sub substrate, it not realistic to always assume full trust in all brokers [16, 18]. This is because malicious brokers may exist in an infrastructure which spans different administrative domains. Basically there are two main approaches to define trust in Pub/Sub brokers. The first approach is to assume that these brokers have different levels of trustworthiness [18]. This approach allows an event owner to use a trusted sub-graph of the whole Pub/Sub network graph to serve its own event type. The second approach is to assume completely entrusted broker network [16]. This approach is supposed to work only with the selective attribute based encryption proposal, which is reviewed in section 2.4.2.

a. Partially Trusted Broker Network

In this approach, the authorization of certain brokers is restricted so that they are not allowed to serve certain event types [18]. Such restriction is specified using the associated access control policy rules with this event type, which is defined by the event type owner or a separate policy manager entity. To specify which brokers are allowed to serve a given event along its way from a publisher to a subscriber; trust in brokers is distributed through the Pub/Sub network [18].

It is noticed that a broker can judge only neighbouring brokers, e.g. their performance. By utilizing this notice, a web of trust is build to span the sub-graph of the Pub/Sub brokers' network with represents the authorized brokers to serve a given event type. In particular, certificate chains are used to form this web of trust. Each broker is assumed to hold what is called an appointment certificate.

An event type owner signs the certificates of the brokers which it trusts, and which are connected to it. In turn, these brokers sign the certificates of their neighbour brokers, which they trust, and so on. In addition, a publisher/subscriber has a trusted root certificate for event type owners, so that it can verify whether a local broker is authorized to publish/deliver a given event type from/to it.

b. Totally Entrusted Broker Network

This approach assumes that a publisher/subscriber may not trust the broker network [16]. The main notice which this approach focuses on is that publisher/subscriber only deals with border brokers, i.e., no direct contact among publishers and subscribers. This notice is utilized so that no security associations between publishers and subscribers are required.

In particular, a technique called proxy re-encryption is used to transform packages from being encrypted using publishers' public key into being encrypted with subscribers' public key [16]. This is done while assumes adversary access to the traffic within the broker network as well as the traffic between publisher and subscribers and the Pub/Sub network.

From trust point of view, an event type owner can allocate the role of proxy re-encryption [16], and the role of key management, to only a trusted entity [14]. This party can be a trusted third party or a member in the owner group of this event type. This is done for each individual event type, and this entity is responsible for generating encryption keys, refreshing them when necessary, and delivering them to all concerned entities.

Furthermore, the entity which performs the key management functionality may consist of a group of brokers [14]. This can be done to achieve this functionality in a distributed fashion of shared secret [16]. And in a topologically dynamic

Pub/Sub substrate, in a ubiquitous environment, such key group may occasionally change. In turn, a higher level of trust must be supported.

In particular, when a new node wants to join the key managers group, then a responsible node, which have the highest degree of trust and which is referred to as the key group manager must check if that new node is authorized to join the group. This is important to keep trust in the key managers group [14].

In this section, we reviewed how the trust issue is addressed in Pub/Sub infrastructures. Firstly, we started by reviewing the special feature of dynamism in ubiquitous Pub/Sub infrastructures, which complicates handling trust in these environments. Then, we reviewed a significant proposal which links trust to authorization of event access in Pub/Sub infrastructures [14]. It starts by assuming a very tight trust scope, in only the event type owner and ends by widening it to the whole Pub/Sub broker network. In addition, it defines the means by which authorized entities can be discriminated.

Finally we reviewed two proposals with more limited trust assumptions in the broker network. From the perspective of this thesis, we assume a fully trusted Pub/Sub brokers' network. In addition, the generated keys by our proposed scheme can also used for selective attribute based encryption to support totally entrusted Pub/Sub brokers' network. Also, we assume the existence of some access control entity which is responsible for authenticating interacting entities for authorized access. The details of this issue r out of the scope of this thesis.

2.5 Summary: *The Scope of Thesis*

In this chapter, we reviewed the Pub/Sub middleware. We reviewed SIENA, a content based Pub/Sub event service which we start our work from. In addition, we reviewed the ubiquitous environment and its requirements at the middleware level. And finally, we viewed these requirements from the perspective of Pub/Sub infrastructures.

From the perspective of this thesis, SIENA is the starting point of our work; we reviewed it in section 2.2.4, after a very essential introduction to Pub/Sub in the rest of section 2.2. Also, the characteristics of ubiquitous environments are very essential for our work. These characteristics have been aggregated by the volatility principle, which we reviewed in section 2.3.1. In addition, we focus on providing privacy aware encryption in ubiquitous Pub/Sub infrastructures. This is to comply with the two emphasized requirements of ubiquitous middleware which we reviewed in section 2.3.2.

Also, we reviewed how current Pub/Sub proposals address users' privacy. In particular, in section 2.4.1, we reviewed how privacy rules can be described and how they are used to control event dissemination. At the beginning of our proposal, in section 4.1, we use this review to make design assumptions. Also, this review is behind a starting point of our proposal, which is viewing privacy variables in terms of hierarchies. However, the way of defining and evaluating privacy rules are out of the scope of this thesis.

Then, in section 2.4.2, we reviewed two main approaches for encrypting events for content based Pub/Sub, namely, event based encryption and attribute based encryption, our proposal follows the first one. Finally, in section 2.4.3, we reviewed the issue of trust in Pub/Sub infrastructures. We assume a fully trusted Pub/Sub brokers network. Also, our key establishment proposal can be used for selective attribute based encryptions in totally entrusted brokers' network. However, providing access control and providing certification for the purpose of trust verification are out of the scope of this thesis.

Chapter 3

Secure Group Communication in Publish/Subscribe Infrastructures

In this chapter, we start by defining the problem of this thesis as a secure group key establishment problem at the Pub/Sub last mile. And for an efficient solution of this problem, we review the possible ways of efficient group key distribution, considering the volatility of the ubiquitous environment and its large size subscribers groups. Secondly, we define another perspective of securing delivered events, which is the concept of security levels. This concept introduces the requirement of encrypting the same event using different keys to reflect different security levels.

Thirdly, we review a set of design guides which should be considered for designing group key establishment schemes in ubiquitous environments. Combined with this review, we specify which of these factors we consider, and how we plan to consider them, and which we do not. Finally, we review a proposal of clustered group key establishment, which is the main architectural approach behind our proposal.

3.1 Group Communication in Publish/Subscribe Infrastructures

Client-server applications employ point-to-point data delivery, which is known as *unicasting*. In contrast, when it's required to deliver data from one or more sender(s) to many receivers, *multicasting* is used as an underlying efficient communication substrate. This type of communication is referred to as *group communication* [19]. For Pub/Sub based interaction in ubiquitous environments, group communication exists when data is disseminated from a broker to neighbour brokers. Also, it exists at the Pub/Sub last mile, where a terminal/local broker propagates a received event to interested subscribers groups.

Multicasting at the Pub/Sub last mile differs from multicasting inside the Pub/Sub brokers' network. In particular, the size of the receiver group and the dynamism of its membership are the reasons of this difference. At the Pub/Sub last mile the number of the members in the receiver group is much higher and the receiver group membership changes in a more frequent rate [20].

Consequently, the problem of securing communication at the Pub/Sub last mile in ubiquitous environments can be defined as a special case of the secure group communication problem [20, 21]. The special considerations are the larger groups' sizes and the dynamic membership, which are under the main focus of this thesis.

3.2 Secure Group Communication at the Publish/Subscribe Last Mile

A trivial solution to secure the event delivery from a Pub/Sub broker to the interested subscribers group may be defined in terms of point to point security, i.e. between the local broker and each client. Accordingly, whenever a subscriber wants to subscriber to an event; the trusted local broker authenticates this subscriber and establishes a session *individual symmetric key* to share with it [20, 19]. Then, the established key is used to for encrypting the subscribed events, which are sent from the broker to the client using unicasting. However, such solution does not scale for dynamic and large size groups of subscribers, which is a typical feature in ubiquitous environments.

Another alternative is to use *broadcast encryption* [20]. This approach requires using a large number of encryption keys for large broadcast messages. In addition, the coalition among unprivileged users can decrypt the information. Moreover, by lowering the number of required keys for long broadcast, it's allowed to a fraction of unprivileged users to be able to decrypt the message. Consequently, both individual encryption and broadcast encryption are not suitable for the purpose of encrypting multicast events messages **[**20**]**. In turn, we may consider using *group key encryption* to perform only one encryption operation on an event before sending it to all subscribers. As a result, multicasting can be easily used to deliver encrypted events instead of using unicasting or broadcasting.

3.2.1 Group Key Establishment for Publish/Subscribe Subscribers Groups

Subscribers can be grouped based on the events they are interested in. These interests are dynamically changing, which makes establishing static secure groups a non-practical solution from the perspective of security management. This is completely not suitable for the very large numbers of subscribers, especially in a case like ubiquitous environments. Dynamic establishment of secure groups is the practical choice for this case [20]. In turn, we have to choose a suitable group key establishment scheme.

Group key establishment protocols can be divided into two main categories viz. *key distribution protocols,* which is mainly a centralized key management scheme, and *key agreement protocols* [20, 22]. For the first category, one party generates or obtains secret keys and transfers them to other parties. For the second category, two or more communicating parties contribute information to jointly establish the shared secret keys. This is done in away which does not allow entities outside the target group to reveal any information about the shared secrets. In general, key agreement protocols have more computational cost than the key distribution schemes. In addition, key agreement protocols are mainly designed for the cases when there is a lack of trusted platform. Contrarily, in Pub/Sub infrastructures, the brokers' network can be used to provide such platform. Consequently, group key distribution schemes are more relevant for our purpose of securing last mile event dissemination [20].

3.2.2 Group Key Distribution and Updating

On the other hand, to keep the subscribers updated with their group key we have to consider the frequent subscribe/unsubscribe operations, i.e. the frequent join/leave operations, by these subscribers, as group members [20, 19]. Updating the group key after every join and leave may result into higher overhead of session establishment than the overhead of unicast session establishment. Even if a trusted local Pub/Sub broker updates a newly joining subscriber with the currently used group key, which is not secure, when a subscriber leaves the subscribers group, a newly created group key must be

communicated by means of unicasting to all other group members. This process does not scale to large size groups.

Consequently, the issue of updating the group key of a subscribers group can be defined as a problem of frequent key changes for a very large group [20, 19], which is a typical case for our volatile and large size groups of interacting entities in ubiquitous environments. We have to consider the scalability of key distribution [23]. The cost of group key establishment is relevant to the group size. To achieve scalability, decoupling of the group size form the costs of the key establishment should be a design requirement for the key establishment scheme.

A possible solution may be defined by decomposing a large group of subscribers into many subgroups and using a hierarchy of security agents to manage the keys different subgroups at different levels of the hierarchy. An example of this approach is reviewed in this section. Even in the case of key establishment by a trusted third party, using hierarchical key distribution architectures is also possible [23]. In our work, this trusted third party is a trusted local Pub/Sub broker. In our future work, we map scalable key distribution network [23] to the Pub/Sub broker network, while in the current version of our work, we consider using another solution, i.e. key hierarchies. An example of such solution is reviewed in this section.

Subgroups Hierarchies for Efficient Group Key Distribution

A typical example of large and highly dynamic networking environment is large size ad hoc networks. With the special scalability requirement in such case, hierarchal arrangement of geographically scattered mobile nodes is proposed as a scalable key management scheme [24]. In particular, the scalability problem is solved by partitioning the communication devices into subgroups. Subgroups are organized into a hierarchy. Among the nodes in a subgroup a leader is elected. A level in the hierarchy is called a tier or a layer. An example hierarchy of security agents is illustrated in figure.6.

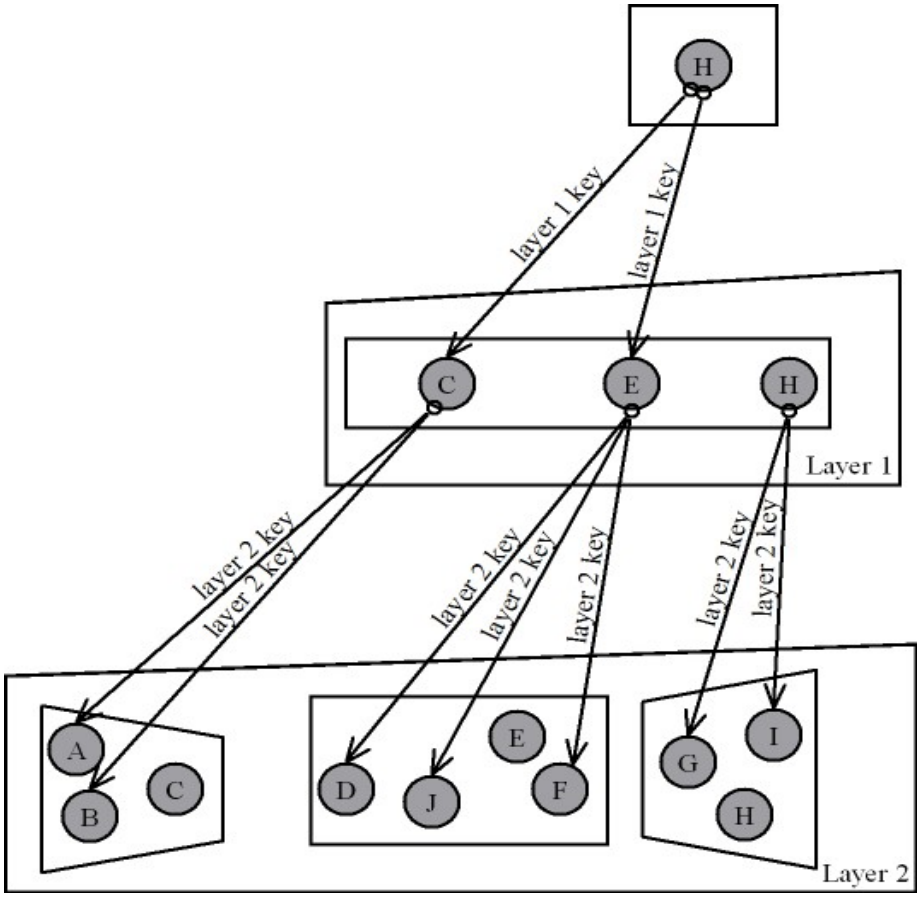

Fig.6. Key Distribution in Subgroups Hierarchy [24]

For each layer, a secret key is defined; this key is hold by the members of the nodes hierarchy which are members of subgroups in that layer [24]. This key is generated by a key server whenever there is a need to refresh it, i.e. a node joins/leaves the layer. In addition, each subgroup has its own subgroup key, which is hold by only the subgroup members. The subgroup leader generates the subgroup key and distributes it to a group member by encrypting it using a secret key which it shares with that particular member.

When a new node joins the group, it is joined to one of the subgroups at the lowest layer of the hierarchy [24]. Contrarily, when a current member node

leaves the group it's removed from all the subgroups at all layers. For both cases, the leader of a subgroup with a membership change obtains a new subgroup key and sends it to its subgroup members, by means of unicasting. Then, for any affected layer, a new layer key is generated and the subgroup leader, at the higher layer, multicasts this key to its subgroup members by encrypting it using the previously generated subgroup key. Such approach is scalable for key distribution to high dynamic large size groups.

From the perspective of this thesis, we consider the possibility of using subgroups hierarchies only as improvements of our current proposal. We mainly consider another approach for efficient group key distribution, i.e. using keys hierarchies. Next, we review this approach.

Keys Hierarchies for Efficient Group Key Distribution

A different approach of efficient key distribution for dynamic and large size groups is defined by generating keys hierarchies [19]. It assumes the existence of a trusted *key server,* in the context of this thesis it can be a trusted local Pub/Sub broker, which generates and securely distributes keys to subscribers. In particular every subscriber has an individual key, which is only shared with the key server. At the same time, a group key is shared by the key server and all the subscribers. This key is used for the secure delivery of events to interested subscribers groups. In addition a key graph is defined. From this key graph, a subscriber is given a set of other keys which exist in the graph along the path from a leaf node, which represents its own individual key, to the root node, which represents the group key. This set of keys is defined as subgroup keys which the subscriber belongs to. These keys are used for scalable and efficient key updates.

An example of a key hierarchy is viewed in figure.7 [19]. When a subscriber represented by node u9 subscribes, i.e. joins the group, the key server generates a new subgroup key, i.e.k_{789}, to replace k_{78} and a new group key, i.e. $k_{1\text{-}9}$, to replace $k_{1\text{-}8}$. Then, the key server distributes $k_{1\text{-}9}$ to subscribers from u_1, u_2 and u_3 (u_4, u_5 and u_6), encrypting it using the sub group key k_{123} (k_{456}). Also it

distributes both k_{1-9} and the k_{789} to subscribers u_7 and u_8 using their old subgroup key u_{78}. Finally, the key server distributes k_{1-9} and k_{789} to u9 by encrypting them using the individual key of u_9. As a result instead of doing thirteen encryption operations in this case, only six operations are required. Such solution should be scalable to large and dynamic subscribers groups in a ubiquitous environment.

From the perspective of this thesis, we use key hierarchies for efficient distribution of generated keys by our proposed scheme. Next, we review how key hierarchies were proposed for efficient group key distribution at Pub/Sub last mile [20].

Keys Hierarchies at the Publish/Subscribe Last Mile

Key hierarchies provide an efficient approach to achieve scalable and secure key distribution for dynamic groups [19, 20]. A trivial suggestion to apply them for secure event multicasting to interested subscribers is to form a key tree for the subscribers group of each particular event [20]. The key tree will be reconstructed for each join/leave event of the subscribers group, a process with a high computational cost.

An alterative approach is to construct a full key tree which represents the whole set of subscribers for all events [20]. When delivering an event, the key tree will be searched to find the smallest subset of intermediate encryption keys which cover all the subscribers interested in this event. The cost of this approach depends on the cost of the search process. A more improved approach is achieved by means of group key caching [20].

To sum up, we notice that key establishment mechanisms mainly depend on the group membership to establish group keys. In this thesis, we minimize this dependency by using a key server for key establishment and key hierarchies for efficient key distributions.

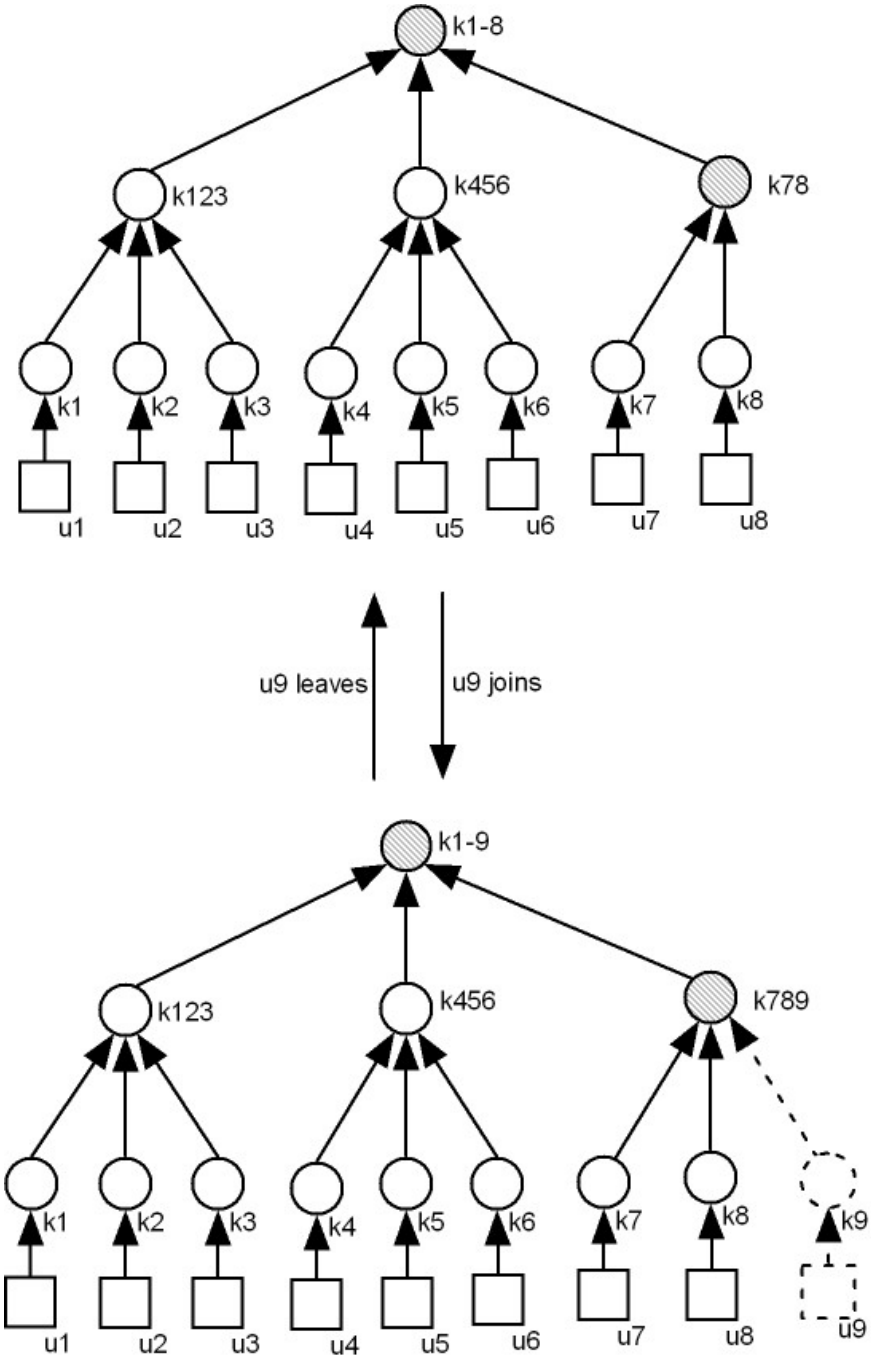

Fig.7. a node joins and leaves a group in a key hierarchy [19]

In the next section we review a very important concept, i.e. security levels. This concept goes beyond just encrypting a specific event for secure event delivery. In particular, it is found that different levels of encryption for the same event are required [25].

3.3 From Security Levels to Privacy Rules

An important perspective of secure group communication is the key generation and distribution for secure inter-group communication. Typically, nodes are divided into multiple groups and traffic exists both within the same group and among different groups [25]. An example of such inter-group communication, which directly applies to ubiquitous environments, is a location service where users' locations are shared with other users.

In such environment, receiver nodes may be divided into groups based on security levels, which are employed to specify the accuracy of delivering location information to them [25]. Consequently, it's required to encrypt the location information at different accuracy levels with different keys to reflect these security levels. In a typical secure inter-group communication scenario, only members in a target group should be able to recover location data at specific security levels which the rest of communicating nodes are prohibited from [25].

In terms of Pub/Sub infrastructures for ubiquitous environments, secure levels can be directly mapped to definitions of access control levels, which in turn can be defined in term of users' privacy policy. As a result, it'll not be enough to accept a subscription from a user only based on his interest. Such subscription must comply with the specified privacy rules fro access the subscribed event.

For the design of our proposed key establishment scheme, we consider the privacy rules of event publishers or event owners. And of course, we have to consider the dynamic and large size subscribers group. This is the major contribution of this thesis. In addition, we consider other factors which affect

the design of our proposal. In the next section, we review these factors and we analyze them from the perspective of our work.

3.4 Key Management for Ubiquitous Publish/Subscribe: *Design Guides*

When designing group key management schemes, several issues affect the design decisions. In general, these issues may include, the security goals of the scheme, the perspective of the users groups, the features of the application environment, the efficiency of the key management process, and how secure is the key management process itself.

In this section, we review a number of influential factors which should be considered for designing key management schemes for Pub/Sub infrastructures in ubiquitous environments. In addition, we specify which factors we consider and which we don't. Also, for the considered factors, we discuss how we plan to address them.

3.4.1 Security Goals for Group Key Management

The key management problem for the subscribers groups, at a local broker in a Pub/Sub network, was defined as a secure group communication problem [20]. Security goals have been defined for a solution to such problem [26]. These goals were stated as follows:

> "*1) Non-group Confidentiality: Clients that were never part of the group should not access any key that can decrypt any multicast data sent to the group;*
>
> *2) Forward Confidentiality: Clients deleted from the group at some time t do not have access to any key used to encrypt multicast data after t, unless they are authorized to join again the group;*
>
> *3) Backward Confidentiality: a user added at time t should not have access to any key used to encrypt multicast*

data before t while the user was not part of the group.

4) Collusion Freedom: no subset of deleted clients should be able to decrypt future group communication, even by sharing the keys they had before their deletion;

*All of these requirements assume that legitimate users do not leak their keys to unauthorized users on purpose.***"** [26].

For our work, we focus on the first three security goals. For the last goal we assume that a trusted Pub/Sub broker, which is local to its clients, is delegated the role of a key manager. This broker is assumed not to disclose enough information about the key establishment process to the subscribers, which limits the chances of collusions attacks. It is also assumed that a subscription, from a client to its local Pub/Sub broker, is encrypted by a shared secret key, i.e. between the client and the local Pub/Sub broker [14].

The above assumptions have the advantage of eliminating the dependency of subscriber nodes on each other to establish their own group key. However, if this is considered as bottle-neck solution, i.e. in terms of overloading the local Pub/Sub brokers, then hierarchical solutions may be employed using subgroups hierarchies of security agents at the local Pub/Sub brokers [24]. An example of this approach was reviewed in section 3.2.2.

3.4.2 User-Centric Group Key Management

In addition to viewing our key establishment problem as a secure group communication problem, we also consider user privacy as our main protected asset by our key establishment proposal. Also, user privacy is considered as a social issue. Social aspects have been studied from the perspective of affecting the design of group key management schemes [27].

For the purpose of our work we consider a number of these issues [27]. The first important issue is the group size, which requires the key establishment

scheme to be completed in the shortest possible time regardless to the group size. This is important because users can't wait for long key establishment times. From the perspective of work, the required time for key establishment is mainly dependent on the key establishment process at Pub/Sub brokers which are local to clients. We consider two core data structures for the key establishment process, each one of them has its own computational complexity. More details can be found in chapter 4.

A second issue is robustness to errors, which is important for ubiquitous assumptions of little, or even non-existing, user awareness [27]. In turn, a key establishment mechanism should assume very limited dependency on users' actions to successfully complete its task. From the perspective of work, the key establishment is done at the side of Pub/Sub brokers which are local to the user. We utilize the inherited clustering of users at their local Pub/Sub brokers as a basis for providing fault tolerance; this is discussed in details in section 3.5.

Another important issue is the structure of the users group. This issue is mainly about selecting a leader node for the key establishment process [27]. In particular, such selection depends on which node the group has trusted as a leader. In our proposal, a local Pub/Sub broker to the clients is used as a key server. In our future work, higher level key managers may be employed. More details can be found in chapter 4 and 5.

A final important issue is the membership flexibility. From the perspective of our work, this is mainly about the frequent change in users' interests, which results into the frequent join/leave rates. We address this issue by employing logical key hierarchies, which have been reviewed in section 3.2.2, in our current proposal. More details can be found in section 4.7.5.

3.4.3 Group Key Management and the *Volatility* Principle

As mentioned in section 2.3.1, a main characteristic of the ubiquitous environments is defined by the volatility principle. This feature should be considered as a main design principle for ubiquitous solutions. Similarly, volatility is a feature of mobile ad-hoc networks, MANETs. Volatility in MANETs has been considered from the perspective of key establishment schemes [28]. Considering the special requirements of MANETs - which include mobility, constrained resources, and the lack of trusted centralized infrastructure, we may learn from its key establishment solutions to influence our solution in the ubiquitous case.

Several requirements for key establishment schemes in MANETs were defined [28]. Basically, a key establishment scheme is required to be lightweight with respect to consuming bandwidth, energy, storage, and computation costs. In addition, volatility requires that the key establishment times to be shorter and the key refreshment events to be more frequent. However, achieving such requirements is difficult in the case of MANETs because of the lack of trusted infrastructure [28], which requires every node in MANETs to be more aware about other nodes in the network for secure key establishment.

For a ubiquitous environment, volatility exists mainly at the clients' side. In contrast to the case in MANETs, a ubiquitous Pub/Sub infrastructure can be used to provide a basis of trust for a key establishment scheme. In addition, neighbour client nodes may not need to be aware of each other to obtain group keys. This is because they interact with the service through their local Pub/Sub brokers.

Finally, for the requirement of short key establishment times, this is defined as a problem of frequent subscribe/unsubscribe of clients at their local Pub/Sub brokers. In other words, it's a frequent join/leave problem in terms of secure group communication. We address this issue by employing logical key hierarchies, which have been reviewed in section 3.2.2**,** in our proposal. This is

also dependent on the core data structure which is used by the local Pub/Sub broker for key establishment, more details can be found in chapter 4.

3.4.4 Encryption for Secure Group Key Distribution

In a ubiquitous environment, existing devices are usually mobile nodes and sensor nodes, which typically interact by means of wireless communication [1, 29]. For the severely constrained resources of such nodes, the complexity of asymmetric encryption is usually considered as quite highly. Consequently, for any type of confidentiality support for such devices, symmetric keys are usually the preferred choice [26]. Such decision is supported by the results of comparing the computational costs of using symmetric and asymmetric encryptions [30].

For the secure group communication problem, it's important to consider the secure distribution of the group keys to the concerned groups of subscribers. For this purpose, the confidentiality of group key update messages must be protected. In turn, we have to choose between symmetric encryption and asymmetric encryption for encrypting the group key updates. To make this decision for a security solution in a ubiquitous environment, we've to consider the resource constrained devices in this environment, viz. mobile and sensor devices.

We may consider the worst case, i.e. sensor nodes. This is just for making very constrained decision; however sensor nodes will not be involved in the secure group communication problem at the Pub/Sub last mile, which is the main problem of this thesis. It has been shown that the performance of some asymmetric encryption algorithms, e.g. elliptic key cryptography, is very close to being a practical choice for these sensor nodes [29].

The advantage of using asymmetric encryption over symmetric encryption, for the purpose of secure group key distribution, is that the private key of a client node can be pre-stored on this node. This is opposite to the case of pair-wise key establishment, e.g. by means of keys pre-distributions or variants of Deffi-

Hellman key exchange [29]. This property enables asymmetric encryption to be widely used as a security bootstrap, which is the case in Internet. However, the complexity of asymmetric cryptography is still being considered as a big disadvantage with respect to applying it to resources constrained devices. Several efforts have been done and are being done for the purpose of reducing the complexity, and the computation costs, of the asymmetric encryption; but more efforts are required [29].

Compared to the case of sensor nodes, mobile nodes have more relaxed resource constrains. In this thesis mobile nodes represent the client nodes in the secure group communication problem of Pub/Sub events delivery. In turn, asymmetric encryption may be suitable for securing group key update messages to these nodes. But this can contradict with the frequent subscribe/unsubscribe events, i.e. frequent join/leave events, in ubiquitous environments.

In particular, asymmetric encryption may be used for the purpose of authentication, which happens only to a newly subscribing node and not to other subscribers, which are current members of the subscribers group. Consequently, symmetric encryption can be used for frequent group key updates, which is a main choice for several group key distribution schemes as we mentioned in section 3.2.2.

3.4.5 Security of Group Key Management

The popular secure group communication mechanisms, e.g. key hierarchies [19], usually consider security issues like forward security and backward security. However, they have been proved to be vulnerable to an attack which targets a legitimate group member to obtain some previously used group keys in addition to the current group key [21, 31]. This contradicts with the belief that such attack can obtain only the currently group key. This attack can be very powerful in environments where it is possible to the attacker to capture legitimate group members, e.g. sensor nodes, ad-hoc nodes, and mobile nodes, which are all typical examples of nodes in a ubiquitous environment.

This attack depends on recording the encrypted traffic to that legitimate node, which includes re-keying events. Then, it makes use of a captured used group key/individual key, which is currently being used for the purpose of re-keying and which can be obtained from the captured node, to decrypt previous re-keying event messages. This way it can disclose past group encryption keys. Immunity against this powerful attack can be achieved by modifying the used group communication key management scheme; however, extra computational costs are required. For the purpose of this thesis, we only consider the common security issues by group communication key management, e.g. forwards security and backward security. This kind of attack is totally out of the scope of our work.

3.4.6 Privacy and Trust in Dynamic Large Scale Communication Infrastructures

In dynamic large scale communication environments, enforcing user's privacy is dependent on the user's trust in the communication infrastructure. This issue is illustrated in very important example environment, which is the cellular access network. This communication environment shares many characteristics with ubiquitous environments, e.g. the large numbers of users and dynamism.

In cellular networks, privacy-aware security underpinnings were investigated [32]. In particular, both the user authentication and the key establishment processes were enhanced to support users' location and identity privacy at different levels of security contexts. Similarly to the performance issues in the ubiquitous environments, privacy-aware security in cellular networks considers performance issues like mobility management signalling, which can have the same position as the frequent join/leave subscribers in a ubiquitous Pub/Sub infrastructure.

Considering the similarity between the ubiquitous environment and the cellular environment, we may learn from the privacy-aware security solution for the cellular environment to influence our security solution for the ubiquitous one. This solution defines a model of a cellular network in terms of three entities,

viz. User Entity, UE, which is typically a mobile node, Home Entity, HE, which manages the UE subscription, location and charging data, and a Serving Entity, SE, which is an access point with respect to the UE [32]. In particular, we may learn how the trust issue is handled to achieve privacy-aware security solution.

From a security trust point of view, the proposal defines three security trust relationships, viz. a long term one between the UE and the HE, another long term trust relationship between the HE and the SE, and finally a transitive medium-term trust between the SE and the UE based on the former two long-term rust relationships. These security trust relationships are discriminated from the privacy trust relationships, where the UE considers both the HE and the SE as just semi-rusted entities [32].

More specifically, privacy levels of the identity and the location of the UE are defined. Firstly, the HE should not be allowed to know the exact location of the UE. Secondly, the SE should not be allowed to know the exact identity of the UE. Finally, an adversary node must not be allowed to any piece of information of the UE [32]. Then, the proposal goes into specific cellular network details to achieve privacy-aware authentication and key agreement protocol, which we are not concerned with them in this thesis.

From the perspective of our work, i.e. Pub/Sub for ubiquitous environments, a UE can be mapped to a publisher/subscriber, and an SE can be mapped to a Pub/Sub broker which is currently local to that publisher/subscriber. In the case of the cellular network, a UE does not fully trust a SE to completely disclose its identity information, e.g. for authentication purposes. A similar case may exist in a Pub/Sub network; a subscriber may not fully trust its local broker from a privacy point of view. As a result, two categories of privacy-aware security solutions may be proposed, viz. one which assumes full trust in the brokers of the Pub/Sub network and another which assumes entrusted brokers.

These trust assumptions affect authentication based access control, which requires user's information to build trust relationships. In particular, for

ubiquitous applications, disclosing user's information to entrusted service providers, i.e. a Pub/Sub broker, contradicts with the privacy of the user [39]. Also, these trust assumptions affects the delegation of group key management, for the purpose of secure event multicasting. For the purpose of our proposal, we consider the case of trusted Pub/Sub brokers; while in our future work we consider the case of entrusted Pub/Sub brokers, including terminal/local brokers.

3.4.7 Secure Collection of Users' Data

In ubiquitous environments, a wide utilization of sensor networks is employed [1]. Their main function is to collect users' data, get their context information, and/or fire events about them. Sensor nodes form the front end of ubiquitous environments. When firing events about collected users' data, sensor nodes should encrypt the events information. A main encryption key establishment scheme in sensor networks is random key pre-distribution [11, 33]. Originally, this scheme is made to efficiently establish unique pair wise keys among sensor nodes in a sensor network.

The key pre-distribution scheme has been improved for deployments in large scale networks with no constrains on the sensors deployment density or distribution [33], which makes it useful for the dynamic large scale ubiquitous environments. Securing the collected users' data by sensor nodes in a ubiquitous is out of the scope of this thesis. Also, we assume that the proposed privacy-aware encryption is done by authorized Pub/Sub brokers, which are local to event publishers, which can be sensor nodes.

In this section, we discussed the considered factors in our design. In the next section we explain the main approach that our proposal follows, namely fault tolerant clustered key management.

3.5 Clustered Group Key Management: *Our Main Approach*

As we mentioned in section 2.3.1, the ubiquitous environment is a volatile and fault prone application environment. Consequently, the main approach of our key establishment proposal must be based on a scheme which is compatible with these two features. In this section, we review a significant fault tolerant clustered group key management proposal [22]. Also, its features are discussed in terms of Pub/Sub interaction.

In our proposal, clients, i.e. Pub/Sub subscribers, indirectly contribute to the key generation process in terms of their subscriptions. At the same time, we propose that the actual key generation process is done by a third trusted party. For the current status of our proposal, a local Pub/Sub broker functions as this trusted third party. For our future work, a special cluster head key manager does the same role. Both of these schemes agree with a recent proposal of clustered key establishment [22].

In particular, even if we use distributed key management schemes, there is also a need for some sort of centralized authority, which at least can control the group membership by means of access control. Two main features arise from following this approach, namely, concurrent key management of client nodes clusters and fault tolerant key management.

3.5.1 Concurrent Group Key Management

To provide a scalable key management scheme, client nodes can be clustered for the purpose of managing the keys of each cluster concurrently while managing the keys of other clusters [22]. For our proposal, such *concurrent key management* is provided at the level of the whole Pub/Sub network. This is achieved by the distribution of the key management of spatially clustered client nodes among their local Pub/Sub brokers, assuming that these brokers are trusted. This is a directly inherited feature from the distributed structure of the Pub/Sub brokers' network. In our future work, concurrent key management is achieved by higher level cluster heads of brokers' clusters.

3.5.2 Fault Tolerant Group Key Management

In addition to scalability, clustering of client nodes is used to achieve *fault tolerance* **[**22**]**. This is done by isolating a cluster of client nodes when a fault, e.g. a communication error, happens inside this cluster. Then fault handling by means of rejoining the cluster members again to the group communication. Such considerations are required for pure key agreement protocols, when there is a big dependency among group members.

In Pub/Sub interaction, client nodes contribute to the key establishment process only by means of their subscriptions. In addition, a natural clustering of the client nodes exists at the local Pub/Sub brokers. As a result, we mainly consider the case of link failure between a client node and its local Pub/Sub broker. The mechanism of detecting such link failure is out of the scope of our proposal; however, we define its handling mechanism as handler of a group leave event with respect to our key management scheme.

3.6 Summary: *Requirements and Primary Solutions*

In this chapter, we defined the problem of secure event multicasting at the Pub/Sub last mile as a secure group key establishment problem, in sections 3.1 and 3.2. The efficient distribution of group keys is a main requirement for our proposal. In section, 3.2.2, we mentioned that we use key hierarchies for efficient key distribution.

In section 3.3, we concluded requirement of encrypting the same event using different keys to reflect different privacy levels. In our proposal, we start our key establishment scheme by viewing privacy dimensions as attributes in Pub/Sub data structures, which can express coverage relations among the values of these dimensions. In section 3.4.1, we concluded the requirements of forward and backward security of the established keys by our proposal. We address these two requirements by considering the dynamics of subscribers groups. In section, 3.4.2, we concluded that users mainly prefer less effort from their side to establish group keys. We address this requirement by depending on their local Pub/Sub brokers of key establishment.

Also, users are concerned with fault tolerant and rapid key establishment process. We address fault tolerance by utilising the already existing clustering of users around their local brokers, as we discussed in section 3.5. For the rapid key establishment, which is also a requirement of the volatile ubiquitous environment as we reviewed in section 3.4.3, we consider the computational costs when we choose the core data structure which we start our key establishment from. More details can be found in chapter 4. Also, a useful extension to our proposal may require assuming entrusted local Pub/Sub brokers. In section 3.4.6, we reviewed how such assumption can be made in a privacy aware manner. Our future work makes a similar assumption and makes use of this reviewed scheme. More details can be found in chapter 5.

In section 3.4.4 we emphasise the suitability of symmetric encryption for secure distribution of encryption keys. However, establishing the key distribution keys is out of the scope of this thesis. Also, the security of the group key management itself and the secure collection of users' data are out of the scope of this thesis, this is discussed in sections 3.4.5 and 3.4.7.

Chapter 4

Privacy-Aware Key Establishment for Ubiquitous Publish/Subscribe Infrastructures

In this chapter, we introduce our key establishment proposal. We design our scheme to be suitable for encrypting events with attributes of privacy dimensions. The assumed usage of this proposal is to secure events distribution at the Pub/Sub last mile, i.e., to subscribers groups. In ubiquitous environments, the subscribers groups are large and dynamic, which imposes more considerations on our proposed key establishment scheme.

In section 2.5, the considerations and assumptions by our proposed scheme were specified. In section 3.6, main requirements were extracted from the reviewed issues in chapter 3. These requirements are addressed in an incremental way through our design attempts in this chapter. Also, in section 3.6, primary solutions were discussed. We use them as starting points in different parts of our proposal. Next, we use the reviewed approaches of defining privacy rules in section 2.4.1 as our first starting point.

4.1 Categorization of Privacy Rules

In section 2.4.1, we mentioned that most common user privacy dimensions in a ubiquitous environment form the following set:

{Identity, Location, Time, Activity, Nearby people} [12]

Also we mentioned that a user can control his information privacy in terms of disclose/not disclose rules or in terms of information resolution. We define these two approaches of privacy policy specifications as two design concepts, which we consider in both our work and its possible future extension.

4.1.1 Binary Privacy Rules

Firstly, a user may define that his information to be disclosed or undisclosed when these information has specific values. For example, he may specify that his location information is not disclosed to another user at specific values of the location dimension. He may also specify this based on other dimensions, e.g. at specific hours of the day if we consider the time dimension or when he is on a meeting with customers with respect to the dimension of nearby people. In this

text, we assume that all the privacy rules are defined in term of "*Allowed-to*" rules [13], i.e., white lists.

4.1.2 Resolution-Based Privacy Rules

Secondly, a user may define that his information to be disclosed only at specific resolution levels of the privacy dimensions. For example, he may specify that his location information is disclosed at the level of the building numbers and not the exact room he is in. The user may also involve other dimensions, e.g. the user may specify different resolution levels of disclosing his location information at different hours of the day, if we consider the time dimension. In this thesis, we do not assume any resolution based privacy rules; however we consider this in our future work. More details can be found in chapter 5.

To realize both categories of privacy rules, we assume that an access control entity has a database of dimensions schema. In this data base, dimension models define the structure of the considered dimensions. In particular, it defines how different values of the privacy dimensions, at different aggregation levels, are related to each other. An example of such dimensions modelling was defined for location based service [34]. In particular, we assume that an access control entity uses the dimension scheme as reference for interpreting the privacy rules specified by the user. In figure.8, we illustrate an example dimension scheme for the location dimension while in figure.9 we illustrate an example dimension scheme for the time dimension.

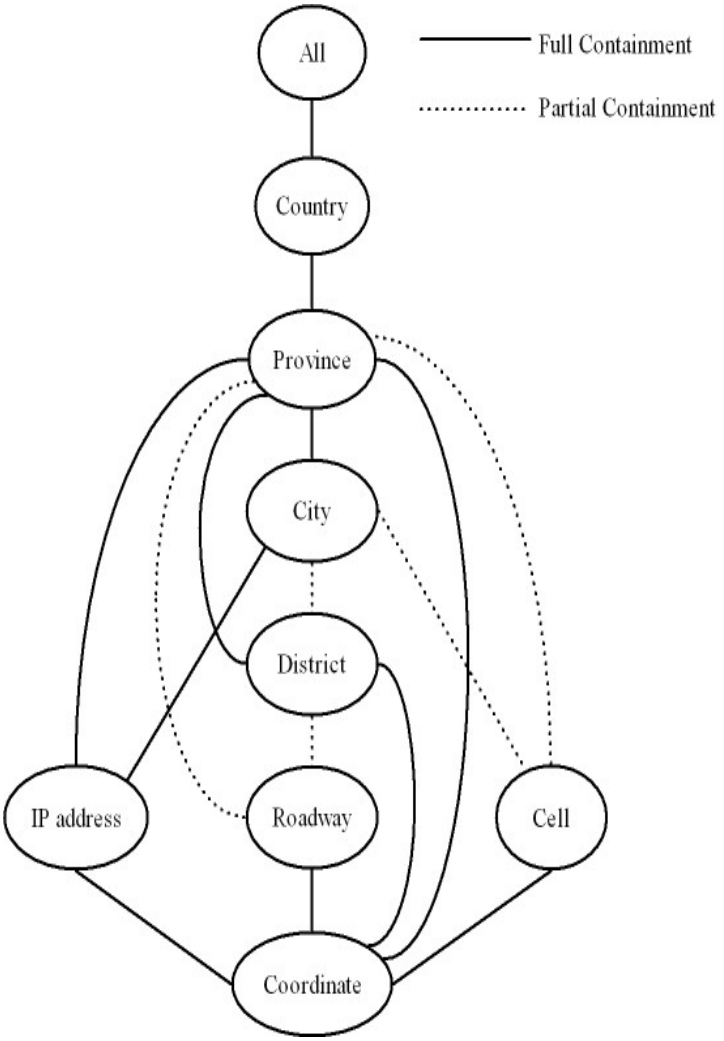

Fig.8. Location scheme [34**]**

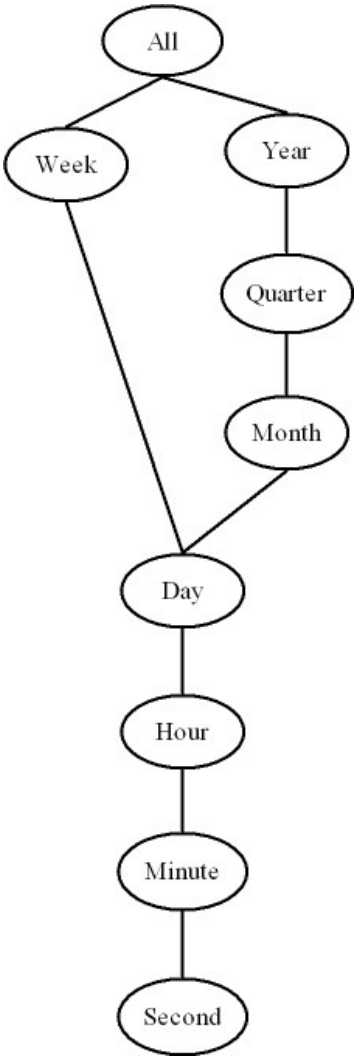

Fig.9. Time scheme [34**]**

4.2 First Data Structure Assumption: *Hierarchies of Privacy Rules*

We assume in section 4.1 that an access control entity uses dimensions schema as reference for interpreting the privacy rules specified by the user. Furthermore, we assume that when we query the access control entity to get a representation of the privacy rules, which were specified by a user in terms of a particular privacy dimension, then a simple hierarchy of this dimension will be returned. If the hierarchy does not have a root, e.g., for *"allowed to any value"* rule, then we assume an imaginary root.

This hierarchy is assumed to include nodes which represent specific values or ranges of the privacy dimension. At each node, there is a set of authorized users who are allowed to access the user's data of this associated value or range. Dimensions hierarchies based on dimensions schema have been defined for location based services [34]. In figure.10, we illustrate an example dimension hierarchy for the location dimension while in figure.11 we illustrate an example dimension hierarchy for the time dimension.

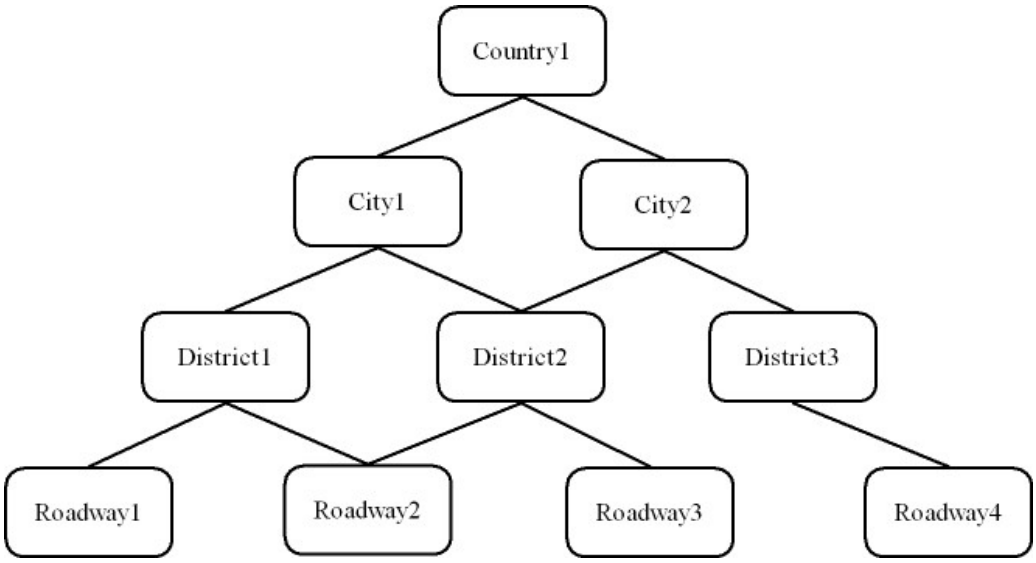

Fig.10. a location dimension hierarchy [34]

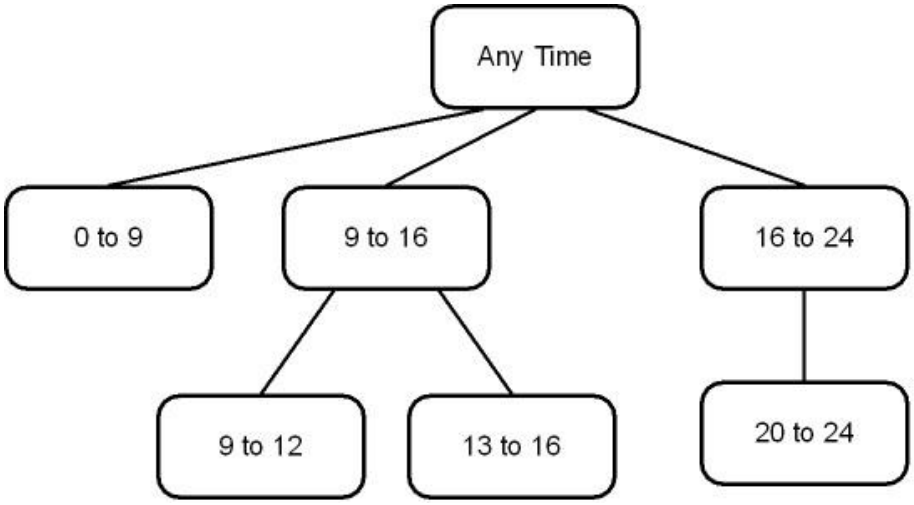

Fig.11. a time dimension hierarchy

4.3 First Design Attempt: *Mapping Dimension Hierarchies to Key Hierarchies*

In this design attempt, we start our design attempt from a recent and significant proposal of attribute-based key establishment [4]. It is proposed for encrypting attributes in Publish/Subscribe events. In that proposal, the relations among the ranges of the values of a given attribute are described as a hierarchy. The way of building an attribute hierarchy is dependent on the attribute's data types.

For example, for an integer attribute, the root of the hierarchy spans its valid full range, while the next level is formed by two nodes which each of them spans half that range. In other words, it is a binary tree representation of the ranges of the attribute [4]. Based on this attribute ranges hierarchy, a key hierarchy is established. In particular, a key is established for each node in the attribute ranges hierarchy. In the next section we explain more about extract

two important design principles from this proposal. In addition, we explain more about building the attribute key hierarchy.

4.3.1 Design Principles

From the above mentioned attribute based key establishment proposal, we can learn the following design principles:

Data-Centric Key Establishment

Key hierarchies can be established in a totally dependent way on the being encrypted data attributes in Publish/ Subscribe events. Viewing the established keys as group keys for subscribers groups to these events, this key establishment approach is totally opposite to the common group-dependent key establishment. If we can totally isolate the key establishment process from the dynamics of subscribers groups, then we can have a volatility-compatible key establishment scheme which is suitable to the dynamics of subscribers groups in ubiquitous environments.

Derivation of Blinded Keys from a Single Parent Key

As illustrated in figure.12, the hierarchy of ranges of a given attribute is represented as a binary tree [4]. For encrypting attribute value for subscribers to the root of the tree, i.e. those who are allowed to read any value of the attribute, it's theoretically possible to use the root key, which does not happen in practice [4]. This root key is referred to as K_{ϕ} . Then, to derive a key to represent the left child of the root node, '0' is concatenated to K_{ϕ} , and a one way function is applied to compute K_0 . This process repeats by concatenating '0' to K_0 to compute K_{00} and '1' to compute K_{01} to go further into the key hierarchy. Similarly, K_1 and its ancestor keys can be computed.

Because a one way function is used to derive child keys, these keys can be referred to as blinded keys [35], because higher child keys can't be used to compute parent keys. This kind of blinded key derivation is suitable for simple privacy dimensions hierarchies, where one-to-many parent-child relationships are guaranteed [36]. And this is the basis of our proposal in the next section.

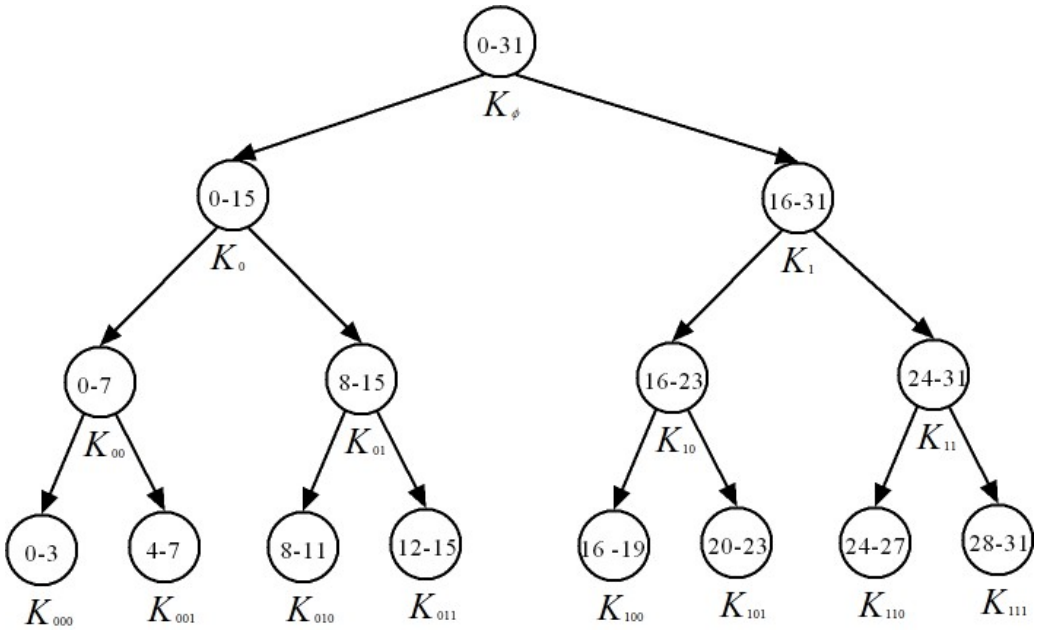

Fig.12. a binary key hierarchy [4]

4.3.2 Privacy-Aware Simple Key Hierarchy

At this point, we can start the proposal of our first design attempt. We start by defining our first proposed design concept.

Dotted Key Index

At this point, we assume using a simple privacy dimension hierarchy. A node in this hierarchy can have only one parent. However, the hierarchy can be represented by a tree of any order n, and not just a binary tree. In turn, we have to review the way we label the key, since concatenating '0'/'1' to the parent key to produce an input to the one way function is only suitable in the case of binary trees. Consequently, we propose the concept of dotted key index.

> ***Design Concept.1***
>
> ***A Dotted key index*** *is a string which represents the position of a key in an n-order tree representation of a given simple key hierarchy. The key index of the encryption key at node n in this tree can be defined as:*

$$keyIndex(n) = i_0 \mid `.' \mid i_1 \mid `.' \mid i_2 \ldots\ldots\ldots\ldots \mid `.' \mid i_r \ldots\ldots\ldots\ldots\ldots \mid `.' \mid i_n$$

, where i_x is the string representation of an integer which represents the position of parent node x among the set of the children nodes of parent node x-1, the node with key index 0 is always the root node, and the operator '|' defines the operation of string concatenation.

The advantage of defining the dotted key index concept is providing a key labelling scheme which is totally independent from the order of the tree which represents the key hierarchy. I.e., i_x can be represented using any number of figures/digits, as opposite to the constraint of using a single binary figure/digit to add a new child in the case of binary tree.

An example of a key hierarchy with key indices labels is illustrated in figure.13. From this figure we can see the root node with key index 0 and its most right child has the key index 4, which in turn has the most right child with key index 4.3.

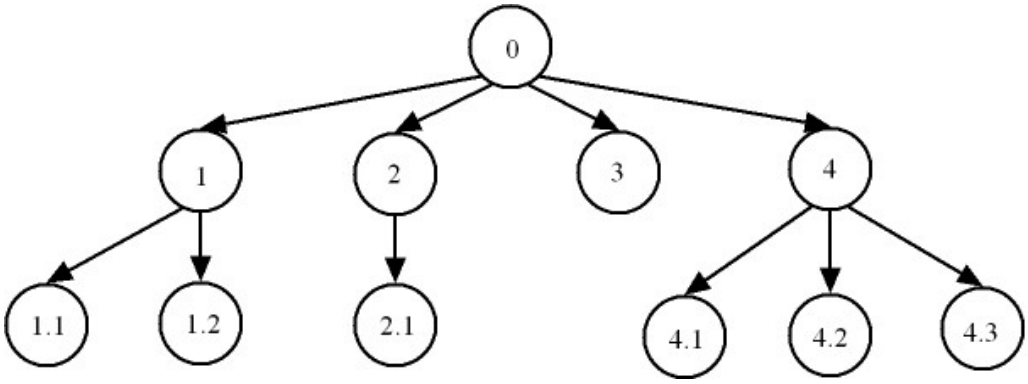

Fig.13. a key hierarchy with dotted key indices

Elementary Operations

According to the above definition, when a node is newly inserted as child node to any other node in a given dimension hierarchy, the key index of this new

child node can be derived from the key index of its parent node using the following procedure:

Procedure.1

Let c be a newly added child node to node p in a simple hierarchy H, which represents the specified privacy rules by user U on the privacy dimension M. Then keyIndex(c) can be computed from keyIndex(p) as follows

1. Increment the numberOfChildren(p) by 1
2. Define keyIndex(c) = keyIndex(p) | '.' | numberOfChildren (p)

, where numberOfChildren(x) represents the number of nodes which have been added to node x as children since the addition of x to the H, and the operator '|' defines the string concatenation operation.

The key index at node *n* is used to compute the encryption key k_n at the same node as follows:

Procedure.2

Let keyIndex(n) be the key index at node n in a simple hierarchy H, which represents the specified privacy rules by user U on the privacy dimension M. Then keyIndex(n) can be used to generate the encryption key at node n, encryptionKey (n) ,as follows:

1. Define keyIndexElements = removeCharacter(keyIndex(n), '.')
2. Define owfInput = encryptionKey(p) | keyIndexElements
3. Compute encryptionKey (n) = OWF(owfInput)

, where removeCharacter(s, c) returns string s without c characters, encryptionKey(p) in the encryption key of node p, the parent node of n, the key of the root node of H is the current priate key of the access

control entity, OWF is a one way function, and the operator '|' defines the string concatenation operation.

Handling Addition of Nodes to Dimension Hierarchy

In terms of the above defined elementary operations, namely Procedure.1 and Procedure.2. We define a procedure to generate key hierarchy from a privacy dimension hierarchy. We assume that it is possible to query an access control entity to return the policy rules of a user from the perspective of a specific privacy dimension *M*. Also, we assume that the query result is represented as a simple hierarchy *H*. We define the procedure of generating a simple key hierarchy, *K* from *H* as follows:

Procedure.3

Let H be a simple hierarchy, which represents the specified privacy rules by user U on the privacy dimension M. When U specifies a new privacy rule which results into inserting a new node c into H, then the privacy-aware simple key hierarchy, K, is modified as follows:

1. If node c is inserted as a child to node p so that c is a leaf node then,

a. Compute keyIndex(c) = Procedure.1(keyIndex(p))

b. Compute encryptionKey(c) = Procedure.2(keyIndex(c))

c. Set the numberOfChildren(c) to integer value 0

2. If node c is inserted as a child to node p so that c is an intermediate node then,

a. Do the same as stated in case 1 of this procedure

b. Compute children(c) as the set which contains every previous child node to p which is now a child node to c

c. For every e $\in$ children(c), call Procedure.4(e)

3. If c is inserted as the root of H then,

a. If there is an assumed imaginary root node, replace it with c.

b. Obtain a new root encryption key

c. Compute children(c) as every node which was at the highest level of H before adding c

c. For every e ∈ children(c), call Procedure.4(e)

, where numberOfChildren(x) represents the number of nodes which have been added to node x as children since the addition of x to the H, and children(x) is the set of children nodes to node x.

In Procedure.3, the creation of a key index for a newly added node to the dimension hierarchy may require to update the keys of the whole key hierarchy, i.e., in case of inserting the new node as a root node. Also, it may require to update a subset of the key hierarchy, i.e., in case of inserting the new node as an intermediate node. We define this updating process as follows:

Procedure.4

Let n be a node in a simple hierarchy H, which represents the specified privacy rules by user U on the privacy dimension M. When the privacy-aware simple key hierarchy K is required to be updated starting from the key index keyIndex(n) then,

1. Set the numberOfChildren attribute of n to integer value 0

2. Compute keyIndex(n) = Procedure.1(keyIndex(n))

3. Compute encryptionKey(n) = Procedure.2(keyIndex(n))

4. Compute children(n) as the set of current linked children nodes to n

5. If children(n) ≠ ϕ , then for every e ∈ children(c), call Procedure.4(e)

, where numberOfChildren(x) represents the number of nodes which have been added to node x as children since the addition of x to the H, and children(x) is the set of children nodes to node x.

After handling the addition of new nodes to the dimension hierarchy we now move to handling the removal of such nodes from the dimension hierarchy.

Handling Removal of Nodes from Dimension Hierarchy

We defined the following procedure to handle the removal of a node from the dimension hierarchy:

> ***Procedure.5***
>
> *Let H be a simple hierarchy, which represents the specified privacy rules by user U on the privacy dimension M, When U specifies a new privacy rule which results into deleting a node n from H, then the privacy-aware simple key hierarchy, K, is modified as follows:*
>
> *1. If node n was a leaf node then, do no thing.*
> *2. If node n was an intermediate node then,*
> *a. Compute children(n) as the set of previous children nodes to n*
> *c. For every e ∈ children(c), call Procedure.4(e)*
> *3. If node n is the root of H then,*
> *a. If there is no node in H which can be used as a root, then create and imaginary root node which represents the "Allowed to any value" rule*
> *b. Obtain a new root encryption key*
> *c. Compute children(n) as every node which is now at the highest level of H after removing n*
> *c. For every e ∈ children(n), call Procedure.4(e)*
>
> *, where children(x) is the set of children nodes to node x.*

The set of above defined procedures are used to build a simple key hierarchy. The procedures utilize a dimension hierarchy which represents the privacy rules specified by a user from the perspective of a particular privacy dimension. In

the next, we explain how the established simple hierarchy can be used in a Publish/Subscribe based privacy-aware application.

4.3.3 Usage

In this section, we explain how the simple key hierarchy mechanism can be used in the context of a privacy-aware Pub/Sub infrastructure. First, we define the interacting entities as follows:

Definition Listing.1

***User Entity**, **UE**: a user who specifies his own privacy rules*

***Access Control Entity**, **ACE**: an entity which maintains the privacy rules of UE and builds simple key hierarchies. An ACE may be implemented in a central or a distributed fashion.*

***Dimension Schema Respiratory, DSR:** a database of dimension schema. A dimension scheme represents the coverage relations of the values of a given privacy dimension. A DSR may be implemented in a central or a distributed fashion.*

***Publish Subscribe Service, Pub/Sub:** a network of Pub/Sub brokers.*

***Local Pub/Sub Brokers, LB:** event routers which are located at the border of the Pub/Sub brokers cloud and which function as access points for the publishers and subscribers to access the Pub/Sub service.*

***Intermediates Pub/Sub Brokers, IB:** event routers which are not directly accessible by Publishers or subscribers.*

***Publisher Entity, PE:** an authorized Pub/Sub publisher entity that may publish events about UE*

***Subscriber Entity, SE:** an authorized Pubs/Sub subscriber entity that may subscribe to receive events about UE.*

Key Hierarchy Maintenance at ACE

In fig.14**,** we explain the sequence of handling of a *UE* request to add/remove a node to/from the simple hierarchy of a given privacy dimensions. First the *UE* submits the request to the *ACE.* In turn, the *ACE* refers to the *DSR* to get the position of the given privacy dimension values with respect to the dimension hierarchy it already has. Then, based on the response from the *DSR*, the *ACE* maintains the dimension and the key hierarchy. Finally, the ACE may or may not acknowledge the UE about the accomplishment of the handling process, which is an open issue.

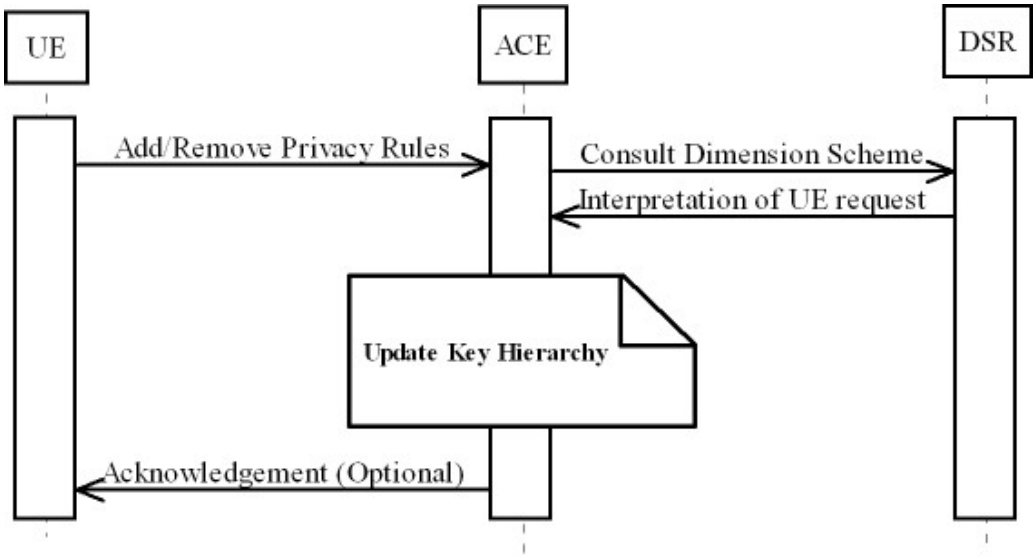

Fig.14. Sequence of handling submitted user privacy rules

Handling Event Subscriptions at Local Brokers

After explaining the process of maintaining the key hierarchy by the *ACE*, we explain the handling of a subscription from *SE* to an event which contains a privacy variable of *UE*. In figure.15, an *SE* sends a message which contains its own access credentials and a subscription request to an event of *UE* to the nearest *LB*. This message is encrypted using a shared secret only between *SE* and *LB*. In turn, *LB* sends an *access-request* message to *ACE* on behalf of the *SE*. This message includes the predicate of the privacy variable of *UE* which *SE* wants to subscribe.

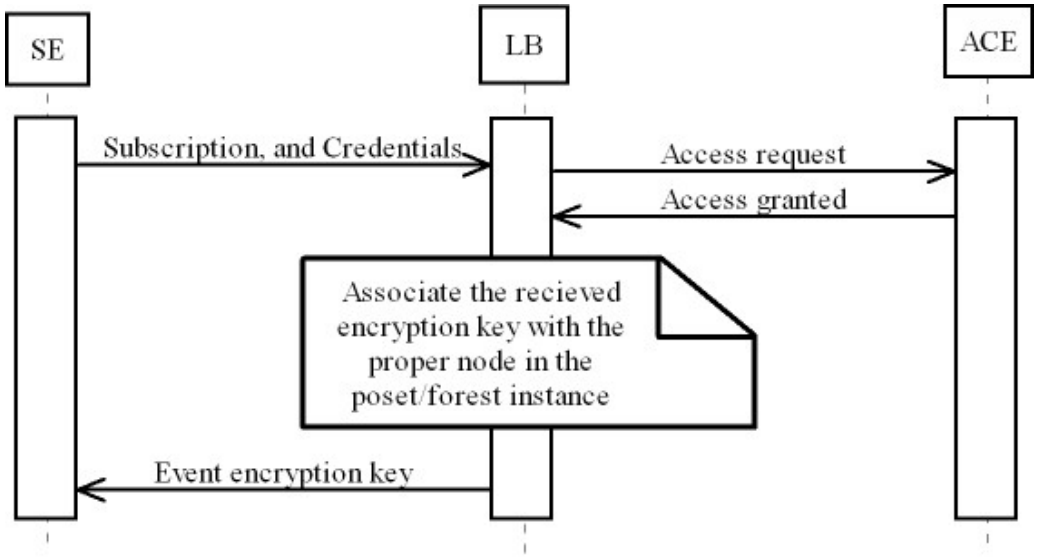

Fig.15. Sequence of handling user's subscription

Assuming that *SE* is authorized to the requested subscription, *ACE* replies to *LB* with *access-granted* message. This message contains the associated encryption key with the lowest level node in the hierarchy of the privacy dimension which covers the predicate in the *access-request* message. It also contains the key index of the included encryption key. Public key cryptography can be used o secure the communication between *ACE* and *LB*.

Finally, the *LB* sends a message which contains the received encryption key and key index to *SE*. This message is encrypted with a shared secret only between *SE* and *LB*. In addition, *LB* stores the received encryption key and key index as associated security data with the node in its poset/forest data structure which contains the subscribed predicate as one of its filters. This is important for the process of privacy-aware secure event multicasting to the set of interested *LB* nodes. We explain this process later in this section.

Group Key Cashing at Local Brokers

From another point of view, the storage of encryption keys and key indices at *LB* nodes can be used as a caching mechanism for the group key. This can be useful in case an *SE* subscribes to an already existing subscription through *LB*. In particular, it enables *LB* to reply to that *SE* with the encryption key and key index associated with this subscription without contacting the *ACE*. However the security of this mechanism is analyzed in section 4.3.4.

Handling Event Subscriptions at Intermediate Brokers

Above, we explained how a subscription request is handled by an *LB*. We have also to consider how *IB* nodes, in the *Pub/Sub* network, should handle forwarded subscriptions from *LB* nodes, with respect to interacting with key hierarchies at the *ACE*. In figure.16, we suggest a possible scenario of this process. For generality, we assume that an *LB* is also a special case of *IB*.

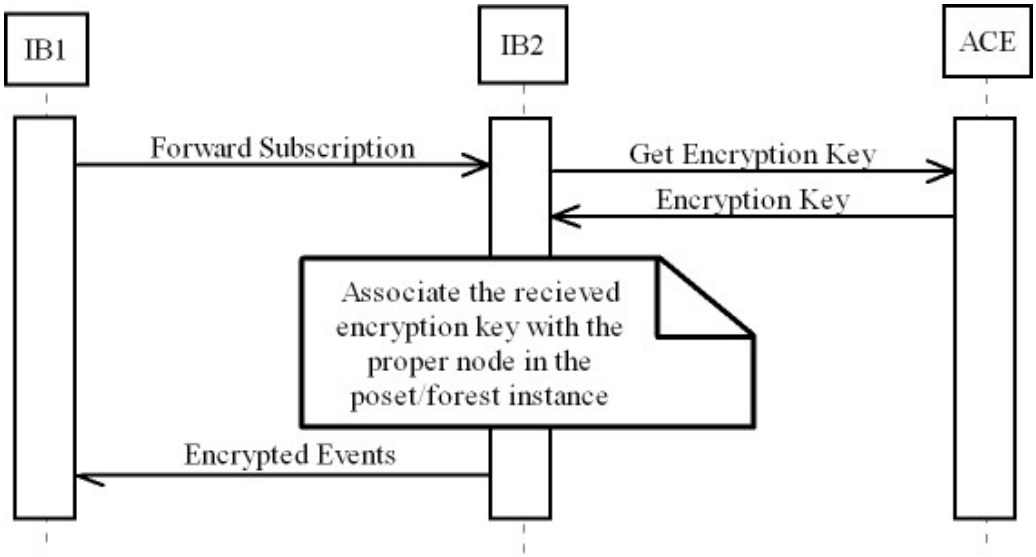

Fig.16. Broker to broker subscription handling and event encryption

In figure.16, an intermediate broker *IB1* forwards a subscription to *IB2*. The forwarding process is assumed to follow the rules of SIENA poset [8, 7]. *IB2* handles the received subscription from *IB1* in the same way as an *LB* handles a received subscription from an *SE*. The only different is that *IB1* does not provide any access credentials to *IB2* to get an *access-granted* response from *ACE*. It is enough that *IB2* gets the proper encryption key and key index from *ACE*. As a rule, *IB2* uses the retrieved key to encrypt any events which match the previously received subscription from, and then it forwards the encrypted event to *IB1*.

This may be utilized for privacy-aware encryption inside the broker network. Basically, this feature can be useful in case of no symmetric encryption underpinning to secure event dissemination among the Pub/Sub brokers. Further investigation of this scenario is not in the scope of this text, however, this is scenario is an inspiration for our future work. Alternatively, we assume the existence of a symmetric encryption underpinning which is used to secure the event dissemination starting from any *LB* which functions as an access point to any *PE*. Consequently, we assume that *IB* nodes follow the rules of SIENA poset [8] for handling received subscription from other brokers.

Privacy-Aware Secure Event Multicasting

At this point, we can explain the privacy-aware encryption at the Pub/Sub last mile. In figure.17, when an *LB* receives an event with a value of the privacy dimension, it matches it against the poset nodes. Then the *LB* uses the associated key index to a matched poset/forest node, to retrieve the associated encryption key. Then, *LB* uses the key to encrypt the privacy dimension attribute in the event.

In turn, *LB* sends the encrypted message along with the key index to all interested *SE* nodes using multicasting. Upon receiving an encrypted event, an *SE* uses the associated key index to retrieve the used encryption key. Finally *SE* uses the retrieved key to decrypt the received event by applying a symmetric encryption process.

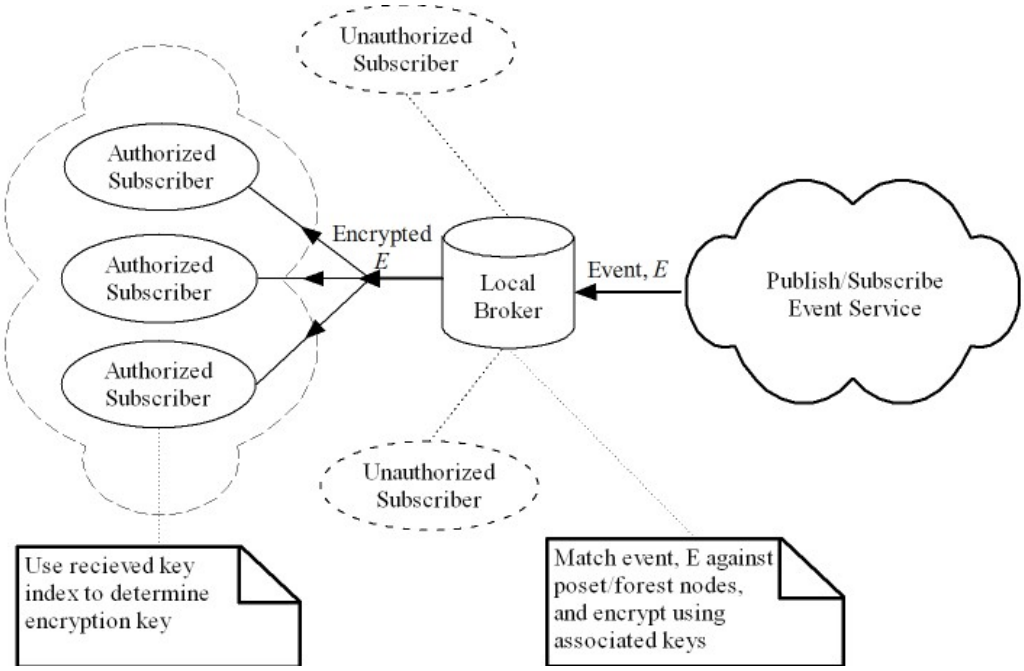

Fig.17. Privacy-Aware Secure Event Multicasting

It's very important to emphasize that a single attribute value in a single event is encrypted as many times as the number of the matching nodes in the poset/forest at the distributing *LB.* At each encryption time, a different encryption key may be used. This means that, a redundancy in associated encryption keys with poset/forest nodes may exist. This is dependent on the consistency between the coverage relations of poset/forest in the LB at one side and the coverage relations of the key hierarchy at the other side, with respect to the privacy dimension of course. We discuss this issue in section 4.3.4; however, we do not investigate it more for this design attempt.

Handling Event Publications

Finally, we explain the handling of publishing events of *UE* by a *PE*. In figure.18, a *PE* sends a message which contains its own access credentials and an event of *UE* to the nearest *LB.* In turn, the *LB* sends an *access-request*

message to *ACE* on behalf of the *PE*. This message includes the value of the privacy variable of *UE* which *PE* wants to publish. Assuming that *PE* is authorized to publish events about *UE*, *ACE* replies to *LB* with *access-granted* message.

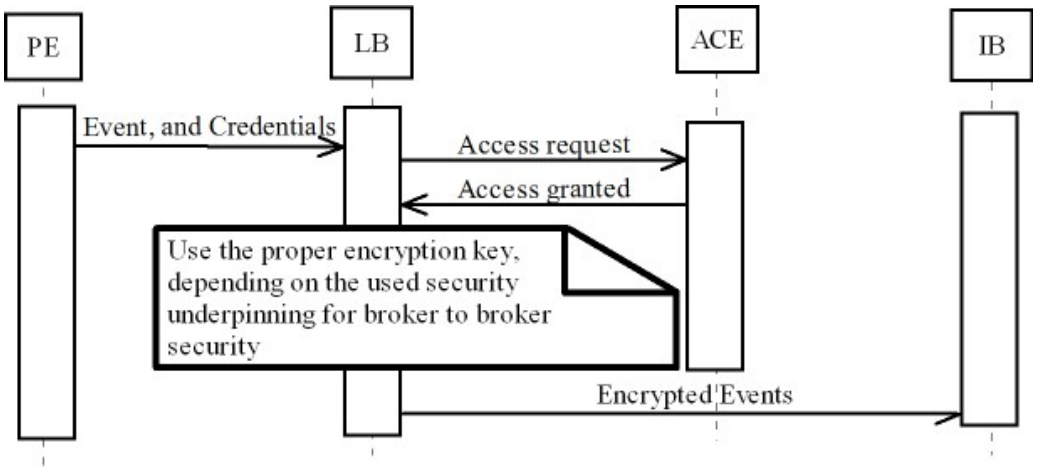

Fig.18. Sequence of handling event publication

At this point there are two possibilities; first if we apply the previously proposed *privacy-aware encryption inside the broker network*, then *LB* uses the associated key index to the poset/forest node, which contains the matched predicate, with the being published event, to retrieve the encryption key. In addition, it is assumed that the neighbour *IB* to this *LB* knows the key index as well as the encryption which the *LB* will use to encrypt the event.

The second possibility is the alternative assumption of the existence of a symmetric encryption underpinning which is used to secure the event dissemination starting from any *LB* which functions as an access to any *PE*. Finally, the *LB* encrypts the privacy variable attribute in the event using the proper encryption key, regardless of the used symmetric encryption underpinning. It's an open issue whether the *LB* should acknowledge the *PE* about the accomplished operation or not.

4.3.4 Analysis

In this section, we analyse our first design attempt. This analysis will move us to conclusions which results into proposing the second design attempt. Mainly we specify our analysis in the following points:

Limitation of Simple Key Hierarchies

In this design attempt, we assumed one parent at maximum for each node in a simple privacy dimension hierarchy. In turn, because the key hierarchy is built based on the dimension hierarchy, the key hierarchy is also a simple hierarchy. However, it can not be always the case that privacy rules are described in term of simple hierarchies. Instead, the key establishment process should start from a data structure which represents the coverage relations among the different values of a privacy dimension in a generalized form.

Of course, one alternative is assuming that an *ACE* is able to represent a given set of privacy rules of a given *UE* as a non-simple dimension hierarchy instead of a simple hierarchy. Such assumption is of course possible according to the categorization of the dimension hierarchies [36]. However, other issues which we describe next in this section suggest the exclusion of this alternative.

The other alternative is to move the key maintenance process to a totally different data structure which represents the coverage relations among the different values of a privacy dimension in a generalized form. SIENA poset is a very significant candidate data structure. In particular, we can view a given privacy variable as an attribute in the filter nodes of poset, which is the starting point of our second design attempt. More details can be found in section 4.6.

Inconsistent Dimension Hierarchy with Subscription Lattice

In section 4.1, we assume that all the privacy rules we consider are defined in terms of "*Allowed-to*" rules [13], i.e., white lists. This means that at least no subscription will be added to the subscription lattice, i.e., poset/forest, at an *LB* unless it's covered by a node in the dimension hierarchy, which represent the privacy rules at the *ACE*. As a result, it's possible that two nodes in a

subscription lattice, e.g. a parent node and its immediate childe, are both covered by the same lowest level node in the dimension hierarchy.

In turn, both of these nodes will receive the same key index from the key hierarchy. Consequently, encrypting the target attribute in an event which matches with both the subscription lattice nodes is done twice, i.e., once for each match result, using the same encryption key. This redundancy, in the encryption process, results from the inconsistency between the dimension hierarchy and the subscription lattice.

One solution is to synchronize the content of these two sides. This is not reasonable because each of them is maintained for a completely different purpose. In particular, this solution requires modifications at the both the *ACE* and the *LB.* Another alterative is to move the key establishment process to the side of subscription lattice, so that the key hierarchy will be a reflection of it. Again, this is the starting point of our second design attempt. More details can be found in section 4.6.

Volatility Compatibility vs. Forward and Backward Security

In section 4.3.3, we assume that the *ACE* maintains the key hierarchy and the *LB* retrieves the proper key indices and encryption keys from the *ACE* to distribute to *SE* nodes. Furthermore, we suggest that the *LB* may even cache key indices and encryption keys for later fast responses to *SE* nodes.

These assumptions have two advantages. Firstly, the key hierarchy maintenance is done at the *ACE* without being aware to any of the subscribe/unsubscribe dynamics at the side of the *LB* nodes. Secondly, the *LB* is able to adapt with frequent changes in the subscribers membership. Both of these advantages are ultimate targets to any design which considers the volatility principle we reviewed in section 2.3.1.

On the other hand, there is a main disadvantage of these assumptions, namely, no support of either forward or backward security, which we define in section

3.4.1. Mainly, this is because the *ACE* is not aware of the group membership dynamics at the side of *LB* nodes. A first possible approach to handle this problem is to refresh the keys in the proper in response to the subscribers' group membership dynamics at the side of *LB* nodes.

But such solution adds more overload to the *ACE*. Particularly, in addition to its main task of enforcing access control, the *ACE* will be aware of the subscribe/unsubscribe dynamics at the side of the *LB* nodes. In turn, a naturally evolving alternative is to move the key establishment process to the *LB* nodes, which this is a starting point of our second design attempt. More details can be found in the next sections 4.5 and 4.6. In the next section we review poset, the core data structure which we base our second attempt on.

4.4 Partially Ordered Set: *Poset*

To be able to forward events based on subscriptions, SIENA defines the filters poset (FP) data structure, which can be thought as the routing table for an event router [8, 7, 37, 38]. An FP stores filters by according to their coverage relations. Most general filters are those filters which are not covered by any other filters in the poset. These filter form is the root set of filters. An FP can be used for matching notifications against filters by traversing it for only matching filters starting from the root set.

From the perspective of our work, FP has a very important feature, namely, *generality*. This makes poset flexible for various filter semantics [38]. In particular, we use this generality combined with the coverage relations among filters to view privacy attributes as filter attributes, regardless of how complex users may specify their privacy policies. However, it's important to note that this generality has been considered as a performance drawback from the perspective of the efficient event routing [38]. This is the reason behind extending our work to consider another more efficient variant of poset, namely, forest [38].

4.4.1 Filter Coverage in Poset

The coverage relation among filters is stated as follows:

Definiton.1

"A filter F1 is said to cover the filter F2, denoted by F1 $\supseteq$ F2, if and only if all the notifications that are matched by F2 are also matched by F1. The $\supseteq$ relation is reflexive and transitive — and defines a partial order" [37, 8].

An event router builds its own FP data structure in response to the subscriptions it receives [37, 8]. For an event router, this FP building process is like the IP routing table building process for an IP router.

4.4.2 Subscription Handling in Poset

The procedure that an event router follows upon receiving a subscription is defined as follows:

Procedure.6

"Subscribers(s) gives the set of subscribers for the given subscription s, and similarly, forwards(s) contains the subset of peers to which s has been sent. A subscription subscribe(X, f) where X is the subscriber and f is the filter representing the subscription proceeds as follows:

1 - A filter f0 is found and f0 $\supseteq$ f $\wedge$ X $\in$ subscribers(f0). Since f for X has already been subscribed by a covering filter the procedure terminates.

2 - An equal filter f0 is found and f0 $\supseteq$ f $\wedge$ f $\supseteq$ f0 $\wedge$ X $\in$ subscribers(f0). In this case, X is added to subscribers(f0). The server removes X from all subscriptions covered by f and subscriptions with no subscribers.

3 - The filter f is placed in the poset between two possibly empty sets: immediate predecessors and immediate successors of f. The filter f is inserted and X is added to subscribers(f). The

> *server removes X from all subscriptions covered by f and subscriptions with no subscribers are also removed."* [37].

Finally, the subscription handling procedure computes the *forwards set* for the newly inserted subscription. This is the set of event routers to which the router will forward the subscription further into the events routers' network [37]. The subscription process procedure has other considerations and optimizations. However, from the perspective of our work, we only focus on the above listed definition, which we will translate later in terms of dynamics of subscribers' groups', which of course affects our key establishment proposal.

In poset, adding a new subscription is considered as a computationally heavy operation [37], because two lookup structures are computed, viz. *predecessors(f)*, which represents the immediate predecessors of filter f, and *successors(f)*, which represents the immediate successors of filter f. Between these two sets, a new subscription is inserted. On the other hand, removing a subscription can be a low computational operation by utilising the already computed *predecessors*(f) and the *successors*(f) data structures [37]. More specifically, the being removed filter is disconnected from its predecessors and successors, then the predecessors and the successors are connecting according to their coverage relations. From the perspective of our work, the process of connecting/disconnecting a subscription filter to/from its predecessors and successor is the most elementary operation that we build our key establishment around it. This is for the purpose of not altering the original poset operations.

4.5 Second Data Structure Assumption: *Reflection of Privacy Rules*

In section 4.3.4, we conclude that we have to start the key establishment from to a totally different data structure which represents the coverage relations among the different values of a privacy dimension in a generalized form. Also, we proposed using SIENA poset to be used because of its generality in representing the coverage relations. Moreover, other points in section 4.3.4

suggest moving the key establishment process to the *LB* nodes, where poset or its variant forest is maintained. This suggestion supports using poset a start for the proposed key establishment mechanism.

In section 4.4.1, we define the coverage relation among filters in poset. Consequently, if we want to use poset as a starting point for key establishment, then we have to map privacy rules to poset. Simply we extend the assumption in section 4.2. In particular, we extend the assumption of a simple hierarchy, which represents the privacy rules from the perspective of a given privacy dimension. We assume that a node in the extend data structure can have more than one parent. This data structure can be referred to as dimension multi-hierarchy [36] or dimension lattice, which directly complies with poset.

From another perspective, we assume that the maintenance and the representation of the privacy rules at the *ACE* side are black-box operations to the *LB* side. In turn we have to define how an *LB* will be aware of the privacy rules at the *ACE* side so that it can represent them in its own poset instance. For this purpose we define the following design concept:

Design Concept.2

Reflected Rules Lattice *is a sub-lattice of the poset lattice at a given Local Broker, LB. This sub-lattice consists of subscription nodes which were received by the LB from a subset of User Entity nodes, Sub-UE, which is defined as follows:*

$$Sub\text{-}UE = \{ue : ue \in authorizedUsers(h), s \supseteq h,\ h \in H, s \in S\}$$

, where H is a simple hierarchy which represents the specified privacy rules by user U on the privacy dimension M, S is the subscription lattice, i.e. the poset instance at LB, and $\supseteq$ *defines the poset coverage relation.*

In the above definition, we mapped the privacy rules which are represented by data structure *H* at the *ACE* side to nodes in the subscription lattice *S*, i.e., poset, at the *LB* nodes' side. In the second and the third design attempts, we start from this design concept, more details can be found in section 4.5, and 4.6.

4.6 Second Design Attempt: *Mapping Dimension Lattices to Key Lattices*

In section 4.5, we extended our data structure assumption from simple dimension hierarchy to dimension lattice be defining the reflective rules lattice. Also, we stated that for this reason and for other considerations, which are stated in section 4.3.4, we start the key establishment from poset. Consequently, the resulting key data structure can be viewed as a key lattice. The main feature of a key lattice is the assumption of the existence of one or more parent node(s) for a single node in the key lattice. In the next section, we define our design principles for the second design attempt based on this assumption.

4.6.1 Design Principles

In section 4.3.1, we defined two design principles that we used for the first design attempt, namely, *data-centric key establishment* and *derivation of blinded keys from a single parent key*. For the second design attempt, we also consider these two design principles. In addition, we define a third design principle which is used to address the assumption of the existence of one or more parent node(s) for a single node in the key lattice.

Mixing of Blinded Keys from Multiple Parent Keys

To define this design principle, we first have to define how we define the privacy levels of the subscribers group in poset as follows:

> ***Design Concept.3***
>
> ***Privacy Levels of Subscribers Groups:*** *According to Design Concept.2, the set of subscribers at node b in poset has a higher privacy level, from the perspective of a given privacy dimension, than that of the set of*

subscribers at node a in poset only and only if b ⊇ a and both a, b contain predicates which are defined in terms of this privacy dimension.

A very significant proposal of group key establishment which has similar assumptions about privacy levels is the *One-Way Function Tree, OFT,* group key establishment [35]. In *OFT,* a binary tree is built to represent a group of communicating entities. The key at a leaf node in this tree represents the private key of one communicating entity. A key is generated for each intermediate node in the tree to be used for secure group communication of the entities at the branching leaf nodes from this intermediate node. The higher the level of the intermediate node, the lower the associated privacy level with its encryption key, and vice versa. *OFT* follows a bottom-up approach from group key generation.

To build the *OFT* key tree, a one way function is applied to the key at each leaf node in the tree to get a blinded version of it. And then, the two blinded keys of two sibling nodes are combined using a combination function to produce their own group key. And the process repeats until we reach the root node. In figure 19, we illustrate an example OFT group key tree, where a group member at node *M* knows only the keys at the black nodes and the blinded keys at the gray nodes [35].

From the perspective of our work, the important part is the ability to mix the keys of two nodes with a higher privacy level to produce the key of a single node with a lower level privacy. We utilize this feature in our second design attempt, which we discuss in the next section.

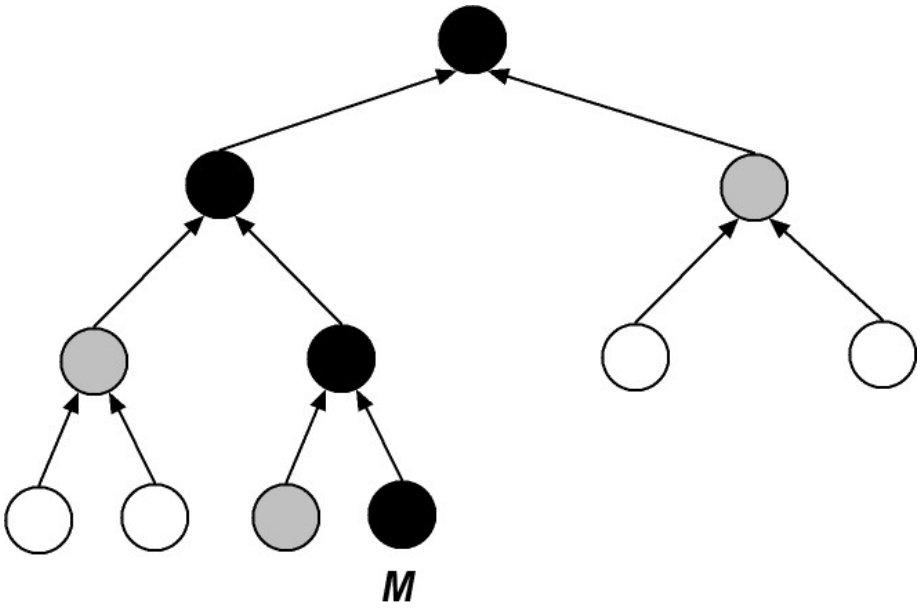

Fig.19. an example OFT group key tree, node M knows the keys at black nodes and the blinded key as the gray nodes [35]

4.6.2 Poset Based Key Establishment

According to Design Concept.3, the higher the level of a subscription node with privacy attribute predicate in poset, the higher the privacy level of its subscribers group, and vice versa. As a result, we may apply the design principles in sections 4.3.1 and 4.6.1 in a top-down approach instead of the bottom-up approach of *OFT* group key establishment [35], which we reviewed in section 4.6.2.

Mixed Dotted Key Index

At this point we extend the design concept of *dotted key index* in section 4.3.2 to be suitable for the case of one or more parent node(s) for a single node in the key lattice. Basically, the extended key index concept must be useful for the purpose of mixing blinded encryption keys from the parent nodes of a node in the resulting key hierarchy to produce the key at this node. We define the concept of *mixed dotted key index* as follows*:*

Design Concept.4

A Mixed Dotted Key Index *at node n in a poset instance is the set of the associated key indices with every possible path from the root node of the poset, which can be an imaginary node, to node n. The mixed dotted key index of the encryption key at node n is defined as follows:*

$$mixedKeyIndex(r) = \{keyIndexElement(r)_I\} = \{keyIndex(r)\} = \{ i_r \}$$

$$, \; mixedKeyIndexElement(n)_p = \{e \mid `.' \mid i_n : e \in mixedKeyIndex(p)\}$$

$$, \; mixedKeyIndex(n) = mixedKeyIndexElement(n)_{p0} \cup mixedKeyIndexElement(n)_{p1} \ldots \cup mixedKeyIndexElement(n)_{pm}$$

, where $r \in rootSet$, the set of uncovered nodes in the poset, i_r is the string representation of the integer value of the position of node r among the children of the poset root node, which may be any imaginary node, i_n the position f node n among the children of its predecessor p, $mixedKeyIndexElement(n)_p$ is the set of inherited key indices by node n from any predecessor node p, keyIndex(x) is defined for any node x by Design Concept.1 as the dotted key index, m is the size of the predecessor set of node n. The node label with mixed key index 0 is always the root node, and the operator '|' defines the operation of string concatenation.

The advantage of defining the mixed key index is to provide a key labelling scheme which is compatible with the complete coverage relations in poset. It is very important to emphasize that the poset based key establishment using this mixed key index concept results into a unique key for each subscription node in poset. This feature is inherited from the uniqueness of the parent groups at each

of these nodes. An example of a key lattice with mixed key indices is illustrated in figure.20.

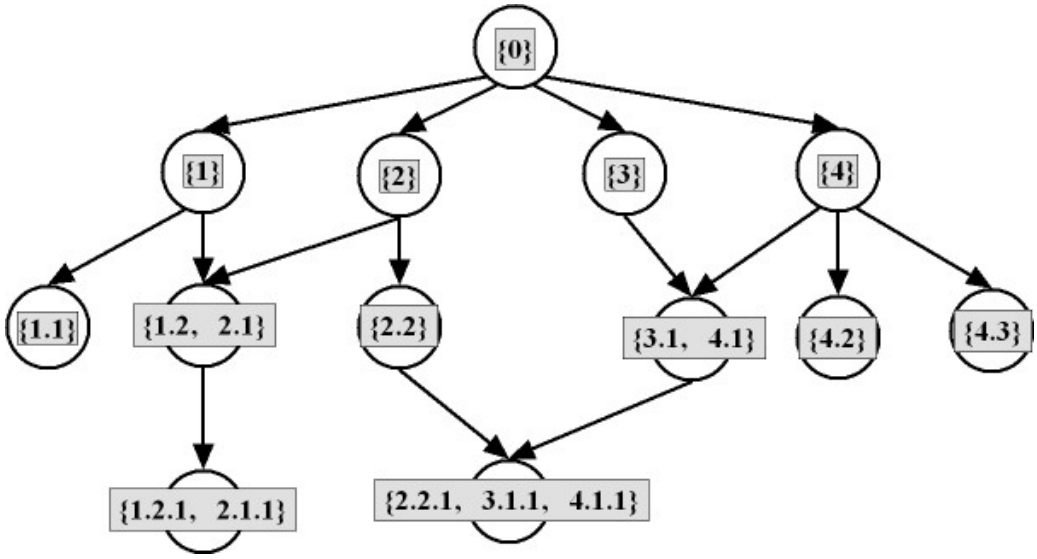

Fig.20. a key lattice with mixed dotted key indices

Elementary Operations

In section 4.4.2, we reviewed the procedure of subscription handling in poset. At the end of that review, we mentioned that we build our poset-based key establishment scheme around the most elementary operation of connecting/disconnecting a subscription filter to/from its predecessors/successors. This is for the purpose of not altering the original poset operations. We define the effects of these connect/disconnect operations on the elements of the dotted key index in terms of the following elementary operations.

First we define a procedure which generates a mixed key index element, for a newly connected child node, from the mixed key index, of a parent node. The procedure adds the position of the newly connected node, among its own children, to all the elements in the key index of the parent node. The procedure is defined as follows:

Procedure.7

Let p be a node in a poset instance, and let n be a newly connected successor node to p, the procedure for generating mixedKeyIndexElement(n) $_p$ *as defined in Design Concept.4 can be descried as follows:*

1. Increment the numberOfChildren(p) by 1
2. For every e $\in$ *mixedKeyIndex(p):*
a. generate ne = e | '.' | numberOfChildren(p)
b. add ne to the set mixedKeyIndexElement(n) $_p$
3. Compute mixedKeyIndex(n) = mixedKeyIndex(n) $\cup$ *mixedKeyIndexElement(n)* $_p$
4. For every c $\in$ *successors(n), call Procedure.8(c)*

, where numberOfChildren(x) represents the number of nodes which have been added to node x as children since the addition of x to the poset instance and successors(x) is the set of successor nodes to node x in the poset instance.

Secondly, we define the procedure which is called to update all the key indices of the successor nodes to node *n* in *Procedure.7:*

Procedure.8

Let c be the set of successor nodes to node n in the poset. Once a new key element in added to the key index of node n, then for an $e \in c$*:*

1. Call Procedure.7(n ,e)
2. If successors(e) $\neq \phi$ *, then call Procedure.8(e)*

, where successors(x) is the set of successor nodes to node x in the poset instance. And n is the parent node in Procedure.7 and e is the child node in it.

Thirdly, we define the procedure which is called to remove a key index element from a given node in response to disconnecting it from a predecessor node. The procedure is defined as follows:

Procedure.9

Let p be a node in a poset instance, and let n be a connected successor node to p, the procedure for removing the set of key elements $mixedKeyIndexElement(n)_p$ *from the mixedKeyIndex(n) can be descried as follows:*

1. Compute mixedKeyIndex(n) = mixedKeyIndex(n) - $mixedKeyIndexElement(n)_p$

2. For every $e \in predecessors(n) - \{p\}$*, call Procedure.7(e ,n)*

, where predecessors(x) is the set of predecessor nodes to node x in the poset instance. And e is the parent node in Procedure.7 and n is the child node in it.

Finally, we define the operation here of generating an encryption key from a mixed key index as follows:

Procedure.10

Let mixedKeyIndex(n) be the mixed key index of a node n in a poset instance, then the process of generating encryptionKey(n) can be defined as follows:

1. For every $mixedKeyIndexElement(n)_{px} \subset mixedKeyIndex\ (p)$

a. For every $e \in mixedKeyIndexElement(n)_{px}$

Compute u = u | removeCharacter(e, '.')

b. Compute owfInput = $encryptionKey(p_x)$ *| u*

c. Compute blindedKey = OWF(owfInput)

d. Compute keyComponets = keyComponets $\cup \{blindedKey\}$

2. For every $b \in keyComponets$

Compute encryptionKey(n) =
mixFunction(encryptionKey(n), b)

, where removeCharacter(s, c) returns the string s without the character c, '|' defines the string concatenation operation, OWF is a one way function, and mixFunction(x, y) is a mixing function, x XOR y is a possible suggestion for it.

After defining this set of elementary operations, we move to the definitions of higher level operations around the connect/disconnect operations of poset.

Handling Connecting/Disconnecting Poset Nodes

In poset, the two ways connection may connect two nodes of a node to another node, i.e., the first node is an immediate predecessor to the second node, and the second node is an immediate successor to the first node. From the perspective of key establishment, this can be viewed as a one way connection. As a result, it's enough that a key establishment process is done at the time of establishing only one direction of this two ways connection.

To keep the key establishment scheme at the lowest possible level, we use the above defined elementary operations when a node connects/disconnects to/from a predecessor node and when a node connects/disconnects to/from a successor node. However, because it is enough to establish a key for only one of these two cases, we leave the performing of the key establishment is to be decided at run time.

In particular, if the upward connection, i.e., from the successor node to the predecessor node, is done first, then the key establishment will be done at that time. In such case, no key establishment process will be done when the downward connection is established, i.e., from the predecessor node to the successor node.

Based on this logic, we define three procedures, which are called for a predecessor node or a successor node without any changes. The first one defines the connection process, the second one defines the disconnection process, and the last one defines the clearing process, i.e., the clearing of the whole set of successors for a predecessor node and the whole set of predecessors a successor node. We define the first procedure as follows:

Procedure.11

Let p be a node in a poset instance, and let n be a successor node to p which is being connected to p, the procedure for updating the mixedKeyIndex (n), which is defined by Design Concept.4, can be descried as follows:

$$\textit{If } mixedKeyIndexElement(n)_p \not\subset mixedKeyIndex(n) \textit{ , then}$$
$$\textit{call Procedure.7(p ,n)}$$

, where p is the parent node in Procedure.7 and n is the child node.

Secondly, we define the procedure for handling the disconnecting of two nodes as follows:

Procedure.12

Let p be a node in a poset instance, and let n be a connected successor node to p which is being disconnected from p, the procedure for updating the mixedKeyIndex(n) which is defined by Design Concept.4, can be descried as follows:

$$\textit{If } mixedKeyIndexElement(n)_p \subseteq mixedKeyIndex(n) \textit{ , then}$$
$$\textit{call Procedure9(p ,n)}$$

, where p is the parent node in Procedure.9 and n is the child node.

Finally, we define the procedure for handling the clearing the whole set of key elements which are inherited from/to all predecessors/successors:

Procedure.13

Let n be a node in a poset instance, if node n is to be completely removed from its current position in poset, then:

1. *If successors(n) $\neq \phi$, then*
   , for every e $\in$ successors (n), call Procedure.9(n ,e)
2. *If predecessors(n) $\neq \phi$, then*
   , for every e $\in$ predecessors (n), call Procedure.9(e ,n)

, where the first passed node to Procedure.9 is the parent node and the second passed node is the child node in it.

In addition, we define how the last defined procedures are viewed from the perspective of the highest level operations in poset, namely, the addition of a new subscription and the deletion of an already existing one. This perspective is important for the purpose of signalling subscribers at a Pub/Sub broker, which maintains the considered poset instance in all defined operations in this section.

For this purpose we define the concept of *Atomic Update Round* of the poset based key lattice:

Design Concept.5

An Atomic Update Round *is defined by a starting point and an ending point. The starting point is the start of a subscription addition/deletion operation to/from a poset/forest instance. The ending point is the end of a subscription addition/deletion operation to/from a poset/ forest instance.*

During the update round, the poset/forest based key lattice is updated, in response to the dynamics of the poset/forest data structure. The round is atomic in the sense that no subscriber is updated with any key index or encryption key during it. Signalling is only done after the end of the round.

This concept adds to the isolation of the key lattice maintenance from the poset operations. This isolation was initially achieved by defining the key maintenance operations at the lowest level operations in the poset data structure. And the definition of this concept isolates the signalling of subscribers about their key indices and encryption keys from the details of the highest level operations of poset.

At this point, we have to explain how the signalling is handled by our poset based key establishment. For this purpose, we define the concept of the set of the to-be-signalled subscribers as follows:

Design Concept.6
The Set of To-Be-Signalled Subscribers *is defined as follows:*

$$signallingSet = subscribers(n_0) \cup subscribers(n_1) \ldots \cup subscribers(n_r) \ldots \cup subscribers(n_n)$$

, where n_x is a node in a poset instance with a modified mixed key index and/or a modified subscribers group, after a single atomic update round. And subscribers(n_x) is the subscribers group to node n_x.

In the above definition, it is important to emphasize it considers any node in the poset with a modified subscribers group but not a modified mixed key index. This is important for the support of forward and backward security. However, it conflicts with the compatibility with the volatility principle; this is discussed in sections 4.6.4 and 4.6.5.

To be able to apply the above defined design concept, we define two procedures. First, we define a procedure to update the keys for the nodes with modified subscribers groups but not modified mixed key indices. We refer to this process as *the pre-signalling modification*. We define this procedure as follows:

Procedure.14

Let n be a node in a poset instance with a modified subscribers group and not a modified mixed key index after a single atomic update round, then the process of ***Pre-Signalling Modification*** *for n is defined as follows:*

For every e ∈ predecessors(n), call Procedure.7(e ,n)

, where predecessors(n) is the set of predecessor nodes to n. And the first passed node to Procedure.7 is the parent node and the second passed node is the child node in it.

Second, we define the procedure of signalling subscribers about updated key indices:

Procedure.15

Let signallingSet be a set of to-be-signalled subscribers as defined in Design Concept.6, then the key updates signalling process for a subscriber s, where s ∈ signallingSet is defined as follows:

1. *Compute signallingSet = signallingSet – {s}*
2. *Set encryptionKey = Procedure10(mixedKeyIndex(subscribedNode(s)))*
3. *Send mixedKeyIndex(subscribedNode(s)) and encryptionKey to s, the sent message is encrypted by a shared secret with s*

At this point, we complete the description of the maintenance of mixed key indices in for a poset instance. In other words, this happens only inside one Pub/Sub broker. In the next section, we define how this key establishment scheme can be used in at the level of a privacy-aware Pub/Sub infrastructure.

4.6.3 Usage

In this section we explain how the poset based key establishment can be applied in the context of a privacy aware Pub/Sub infrastructure. In this section, we use definition listing.101 in section 4.3.3.

Event Subscription and Key Maintenance at Local Brokers

In this point, we explain the handling of a subscription from *SE* to an event which contains a privacy variable of *UE*. In figure.21, an *SE* sends a message which contains its own access credentials and a subscription request to an event of *UE* to the nearest *LB*. This message is encrypted using a shared secret only between *SE* and *LB*. In turn, *LB* sends an *access-request* message to *ACE* on behalf of the *SE*. This message includes the received credentials from *UE*. Assuming that *SE* is authorized to the requested subscription, *ACE* replies to *LB* with *access-granted* message. Public key cryptography can be used o secure the communication between *ACE* and *LB*.

Afterwards, *LB* starts an *Atomic Update Round,* which is defined by Design Concept.5, to update its poset and its key lattice in response to the received subscription from *SE*. Then, *LB* sends to *SE* a message which contains the mixed key index and encryption key which are associated with the subscription node to which *SE* has just subscribed. Also, *LB* signals all the subscribers which the key indices associated with their own subscription nodes have been updated, during *the atomic update round* or during *the pre signalling modification*. The signalling message from *LB* to any *SE* is encrypted with a shared secret only between *SE* and *LB*.

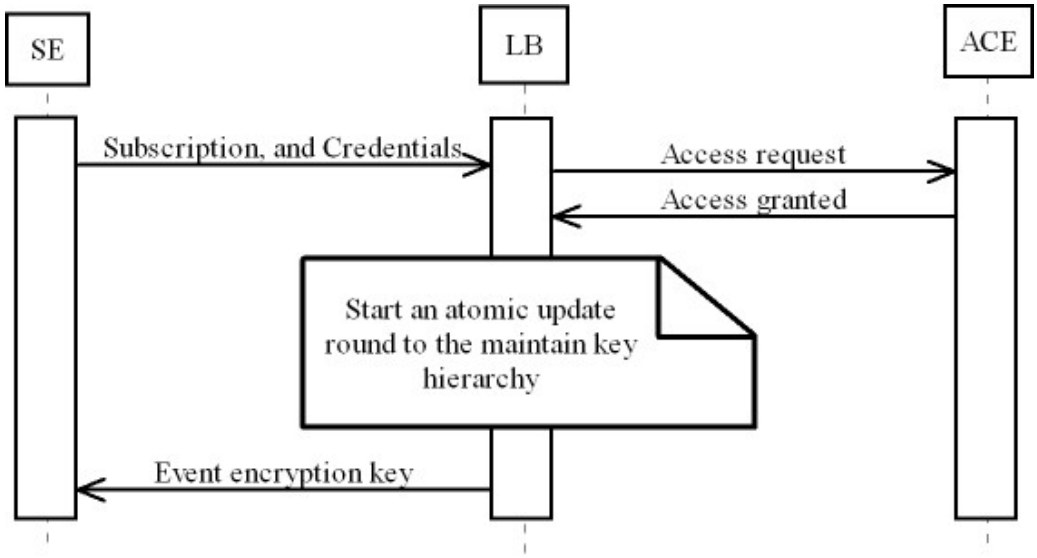

Fig.21. the sequence of handling user's subscription in poset based key establishment

Event Subscription and Key Maintenance at Intermediate Brokers

Above, we explained how a subscription request is handled by an *LB*. We have also to consider how *IB* nodes, in the *Pub/Sub* network, should handle forwarded subscriptions from *LB* nodes, with respect to maintaining poset based key lattice. In figure.22, we suggest a possible scenario of this process. For generality, we assume that an *LB* is a special case of *IB.*

In figure.22, an intermediate broker *IB1* forwards a subscription to *IB2*. The forwarding process is assumed to follow the rules of SIENA poset [8]. *IB2* handles the received subscription from *IB1* in the same way as an *LB* handles a received subscription from an *SE*. The only different is that *IB1* does not provide any access credentials to *IB2* to get an *access-granted* response from *ACE*. As a rule, *IB2* uses the keys in its poset based key lattice to encrypt events which will be forwarded to *IB1*. This may be utilized for privacy-aware encryption inside the broker network in case of no encryption underpinning to secure event dissemination among the Pub/Sub brokers.

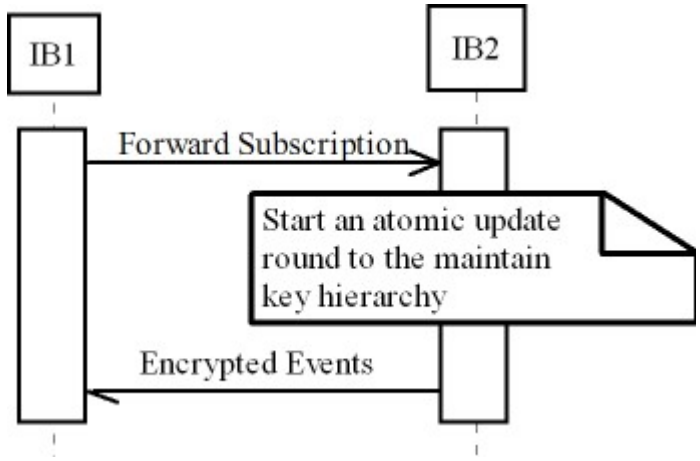

Fig.22. Broker to broker subscription handling and event encryption in poset based key establishment

Privacy-Aware Secure Event Multicasting

At this point, we can explain the privacy-aware encryption at the Pub/Sub last mile. Simply, it is the same process as explained in figure.17 in section 4.3.3, with the replacement of key indices with mixed key indices. However, from the perspective of redundant encryptions issue that we discussed in section 4.3.3, there is a very significant difference, which we explain next.

It's very important to emphasize that a single attribute value in a single event is encrypted as many times as the number of the matching nodes in the poset/forest at the distributing *LB.* At each encryption time, a different encryption key may be used. As opposite to the case of the first design attempt, there is no redundancy in associated encryption keys with poset nodes. The reason is that the key lattice is based on the poset instance itself. In turn, consistency among the key lattice and the subscription lattice naturally exist. This is an advantage which we emphasize in section 4.6.4;

Handling Event Publications

Finally, we explain the handling of publishing events of *UE* by a *PE*. Simply, it is the same process as explained in figure.18 in section 4.3.3, with the

replacement of key indices with mixed key indices. A *PE* sends a message which contains its own access credentials and an event of *UE* to the nearest *LB*. In turn, the *LB* sends an *access-request* message to *ACE* on behalf of the *PE*. This message includes the received credentials from *UE*. Assuming that *PE* is authorized to publish events about *UE, ACE* replies to *LB* with *access-granted* message.

At this point there are two possibilities; first if we apply the previously proposed *privacy-aware encryption inside the broker network*, then *LB* uses the associated mixed key index to a matched poset node, with the being published event, to retrieve the encryption key. Then, *LB* encrypts the privacy in the event and forwards the encrypted event and its mixed key index to the neighbour *IB.* As an *SE* from the perspective of *LB, IB* is able to decrypt the received event. Of course, in case of the existence of a symmetric encryption underpinning, it's used to secure the event dissemination starting from any *LB* which functions as an access point to any *PE.*

4.6.4 Analysis

In this section, we analyse our proposal in the second design attempt. This analysis shows both the advantages and the disadvantages of poset based key establishment mechanism in this attempt. We conclude a significant improvement for it and starting points for the third design attempt. We specify this analysis in the following points:

Generality of Key Lattices and Completeness of Poset

In section 4.3.4, we concluded the requirement of starting the key establishment process from a data structure which represents the coverage relations among the different values of a privacy dimension in a generalized form. In our second design attempt, we proposed the poset based key establishment, starting from SIENA poset, which is featured by it generality in representing coverage relationships.

Also, we defined the concepts of r*eflected privacy rule*, i.e., Design Concept.2, and *privacy levels of subscribers groups*, Design Concept.3. Based on the definition of the coverage relation in poset, in section 4.4, and these two concept designs, we conclude that the proposed poset based key establishment in this design attempt is generalized enough to represent any privacy dimension lattice, which is a main advantage.

On the other hand, the coverage relation in poset is described as a complete one, i.e. a node is connected to every immediate predecessor node or immediate successor node, this results into high computational costs of poset operations. At the same time, our poset based key establishment is built around the most elementary poset operations. As a result, the poset based key establishment inherits this complexity from the poset.

Consequently, if it's possible to build the key establishment scheme around another data structure which provides the same coverage property as that of poset but with lower complexity, then this will be a significant alternative. This conclusion is the starting point of our third design attempt; more details are in section 4.8.

Reflected Dimension Lattice based on Subscription Lattice

In section 4.3.4, we concluded the requirement of moving the key establishment process to the side of subscription lattice, so that the key hierarchy will be a reflection of it. This is mainly to address the problem of redundant encryptions due to the inconsistency between the key hierarchy, of the first design attempt, at the *ACE* side and the subscription lattice at the side of *LB* nodes.

In the second design attempt, we started the key establishment process from the subscription lattice itself, i.e., poset. This totally complies with the above mentioned requirement. In particular, the consistency among the key data structure and the subscription data structure, regardless of their representation, is not an issue any more. This is a main advantage of the proposal in this design attempt.

Forward and Backward Security vs. Volatility Compatibility

In our second design attempt, we proposed that the signalling of subscribers, about their updated key indices, is done in response to the changes of the poset data structure, i.e., after a single *atomic update round*. Also, we defined that the poset based key lattice is modified in response to any changes in the subscribers groups at poset nodes which did not experience any changes in its predecessors or successors sets, i.e., during the last *atomic update round.* In particular, we defined this as the pre-signalling modification.

Finally, we defined of the set of *To-be-signalled* subscribers to contain the subscribers to any node with a modified mixed key index, regardless of the reason of its modifications. It is clear that the proposed scheme considers any possible change in a subscribers group, which is a direct support of forward and backward security.

On the other hand, this feature contradicts with the volatility principle when the mixed index key of a node in the poset is modified while the subscribers group of this node is the same at the start and the end of the *atomic update round.* In section 4.6.5, we propose an extension to our poset based key establishment to be a volatility compatible solution. However, in our third design attempt we consider this issue from the beginning; more details are in section 4.8.

Fault Tolerant Clustered Key Management

In section 3.5.2, we discussed how our planning to move the key establishment process to the local Pub/Sub brokers makes use of clustering subscribers around their own local brokers to achieve fault tolerance from the perspective of the whole Pub/Sub network. In section 4.3.4, we used poset instances at local brokers for the purpose of generating privacy aware keys for subscribers group. This solution complies with the fault tolerance at the Pub/Sub network level.

4.6.5 Volatility Compatible Poset Based Key Establishment

In this section, we propose a modification to our proposal of poset based key establishment to address the issue of volatility compatibility, which we raised in section 4.6.4. We handle this issue at two axes, namely, minimizing the number of key update messages and minimizing the number of distribution encryption operations which are required for the sending the key update messages.

Minimizing the Number of Key Update Messages

For the first purpose, we start by defining the design concept of *Mixed Key Indices Queues:*

> ***Design Concept.7***
>
> ***A Queue of Mixed Key Indices*** *at node n in a poset instance contains a set of mixed key indices. An element of this queue is defined as* $mixedKeyIndex(n)_t$*, where t represents the ascending temporal order of creating the mixed key indices for node n. A consistent key lattice is formed by all last elements in the queues of mixed key indices at all nodes in the poset instance.*

Then, we define the concept of the minimal set of the to-be-signalled subscribers as follows:

> ***Design Concept.8***
>
> ***A Minimal Set of To-Be-Signalled Subscribers*** *is defined as follows:*
>
> $$minSignallingSet = subscribers(n_0) \ldots \cup subscribers(n_r) \ldots \cup subscribers(n_n)$$
>
> *,where* n_x *is a node in a poset instance with a modified mixed key index as well as a modified subscribers group after a single atomic update round. And* $subscribers(n_x)$ *is the set of subscribers to node* n_x *in the poset instance.*

This way, we shrink the definition of the set of the *To-be-signalled* subscribers, that we defined in section 4.6.3, by excluding the subscribers to any node with a modified mixed key index but with the same subscribers group, i.e., at the start and the end of the same atomic update round. It is important to note that the modification of mixed key indices can be due to modifications during the last atomic update round or due to modifications during as the pre-signalling modification; more details are in section 4.6.3.

Finally, we define how the above defined two concepts are used together to provide a volatility compatible key updates signalling. This is defined in the following procedure:

Procedure.16

Let minSignallingSet be a minimal set of to-be-signalled subscribers as defined in Design Concept.8, and let be queueMixedKeyIndices(n) be the queue of mixed key indices of a node n, in a poset instance, as defined in Design Concept.7, then the key updates signalling process for a subscriber s, where $s \in$ *minSignallingSet and* $s \in$ *subscribers(n), defined as follows:*

1. Compute minSignallingSet = minSignallingSet – {s}
2. If subscribers(n) $\cap$ *minSignallingSet =* ϕ*, then*
set newMixedKeyIndex =
removeFirstElement(queueMixedKeyIndices(n))
else
set newMixedKeyIndex =
getFirstElement(queueMixedKeyIndices(n))
3. Set newEncryptionKey = Procedure.10 (newMixedKeyIndex)
4. Send newMixedKeyIndex and newEncryptionKey to s, the update message is encrypted by a shared secret with s

, where subscribers(x) is the set of subscribers to node x in the poset instance, removeFirstElement(Q) returns and removes the first element

from queue Q, and getoveFirstElement(Q) returns the first element of queue Q without removing it.

Minimizing the Number of Encryption Operations for Key Distribution

At this point, we minimize the number of key update messages. Next we minimize the number of encryption keys which are required for the sending the key update messages. For this purpose we address the issue of efficient group key distribution to the minimal set of to-be-signalled subscribers.

In section 3.2.2, we reviewed key hierarchies for efficient group key distribution. We view every node in the poset as a single group key, from the perspective of key hierarchies this key is the key at the top of the hierarchy. Then the local Pub/Sub broker is responsible for maintaining a key hierarchy under this group key, i.e. the root of the hierarchy is not computed. Sub-group keys in this hierarchy change base only on the group dynamics. Sub-group keys are used for efficient distribution of the key at the considered node in the poset instance.

In this section, we introduced an improvement of the poset based key establishment to make it volatility compatible. In the next section we review forest, the core data structure of our third design attempt, which we target to make volatility compatible from the design phase, and with the least storage, and of course computational, costs.

4.7 Poset Derived Forest

Avoiding the computational cost of the add operation in poset was a main motivation to the other proposal that we consider in the context of our work, namely forest [37, 38]. Similar to poset, forest is a generic data structure which stores filters according to their coverage relationships [37, 38]. Forest differs from poset in that each node may have only one immediate predecessor.

4.7.1 Filter Coverage in Forest

A forest is defined as follows:

> ***Definiton.2***
>
> *"Given F as a finite set of filters and $\supseteq$ as a coverage relation, a pair $(F, \supseteq)$ is an X-derived forest, if*
>
> 1. *$F \subset X$, X is a poset, F finite.*
> 2. *For each $a \in F$ there is at most one $b \in F$ for which $b \supseteq a$, i.e. $(F, \supseteq)$ is a forest with the relation $\subseteq$ going from child to parent.*
> 3. *If $a, b \in F$ and $b \supseteq a$, then $a < b$ in X.*
>
> *The set F is called the base set of $(F, \supseteq)$ and the set of all X-derived forests is*
> *denoted DF_X."* [37].

From the forest definition, we can clearly notice that a filter in the forest can't have more than one parent. Consequently, although there may exist a coverage relation between two filters a and c in the case of poset, it is possible that a will not be inserted under c in the forest because a can be inserted in the forest under another filter b. Forest has other definitions and considerations, e.g. maximal poset-derived forest and sibling purity [37], however for the purpose of our work we only focus on the principal idea of forest, which is defined by the above definition.

If we consider using forest for matching events notifications, i.e. by traversing the data structure from starting from the filters at the root of the data structure, this one immediate predecessor constraint dose not conflict with this matching functionality. It's also utilized by the forest subscription addition operation to achieve less computational costs than the addition operation in the case of poset [37, 38]. From the perspective of our work this one parent constrain is utilized for minimal subscribes group key updates, which is an important feature for large sized and highly dynamic subscribes groups.

4.7.2 Subscription Handling in Forest

Forest defines it is addition and deletion operation of a subscription filter as follows:

> ***Procedure.17***
>
> *"Let (F,w) be an X-derived forest. We assume that there is an easy way to get at any node in F from its name. All references to "larger" and "smaller" are to be taken with respect to the relation w on X. We define the following algorithms with inputs F and some x 2 X and output an X-derived forest:*
>
> ***add*** *Set the current level to the roots of F.*
>
> 1. *If x is already in the forest, return without doing anything.*
> 2. *Else if x is incomparable with all roots of the current level, add x as a new singleton tree.*
> 3. *Else if x is larger than some root of the current level, update the current level by making x a root of a new tree with children being all previous roots that are smaller than x.*
> 4. *Else pick a tree whose root is larger than x, set the current level to be this root's children and repeat this procedure from step 2.*
>
> ***del*** *Let C be the set of children of x and S be the set of siblings of x. Then, run add for each of the elements of C, starting with S as the current level. In this an element of C carries the whole subtree rooted at it with the addition"* [2].

Compared to the addition and deletion operations of the poset, these two operations in the forest are very simple and efficient [2]. From the perspective of our work, addition and deletion operations of forest share the lower level operations of connecting and disconnecting nodes in the forest. We view these operations as the most elementary operations than we build our key establishment around them. This is for the purpose of not altering the higher level forest operations, i.e., addition and deletion. In the next section we start our third design attempt.

4.8 Third Design Attempt: *Mapping Dimension Lattices to Key Forests*

In section 4.6.4, we concluded the requirement of starting the key establishment scheme from another data structure which provides the same coverage property as that of poset but with lower computational costs. As we reviewed in section 4.7, the forest has more simple and efficient operations that of forest, while matching and event against its nodes results into the same event forwarding decisions as those of poset. This is because they provide the same coverage relations as reviewed in Definition.1 and Definition.2.

From another perspective, in section 4.6.5, we handled the issue of volatility compatibility for our second design attempt at two axes, namely, minimizing number of key update messages to subscribers nodes and minimizing the number of encryptions which are required for the sending the key update messages. A similar strategy will be followed in our third design attempt but starting from the design phase, as we concluded in section 4.6.4.

In this section we propose the forest based key establishment, our third design attempt. This proposal has very similar design principles, operations, and usage scenario to the previous two attempts. It is designed to be lightweight with respect to computations costs for key establishment and key distribution. Also, it is designed to provide the flexibility of expressing any coverage relations of a given privacy dimension. It assumes the second data structure assumption of section 4.5, i.e. *the reflected rules lattice*.

4.8.1 Design Principles

In section 4.3.1, we defined a set of design principles that we used for the first design attempt, namely, *data-centric key establishment* and *derivation of blinded keys from a single parent key*. For the third design attempt, we only consider these two design principles; there are no additional considerations.

4.8.2 Forest Based Key Establishment

In this section, we start our third design attempt, forest based key establishment. We start by the reasoning behind the conclusion of the central design concept of this proposal, i.e. inconsistent key index.

Event Encryption/Decryption and Consistency of Key Indices

In sections 4.3.3 and 4.6.3, we notice that when an *LB,* as defined *Definition Lisitng.1,* encrypts an event with the associated encryption key with a matched node in poset in order to multicast that event to the set of subscribers at that node. This mans that, a key lattice with consistent key indices, i.e. with the mixed key index at any node is the latest update from its current parent, is not necessary for the this kind of encryption.

In other words, encryption and decryption are independent of the consistency of the key indices. This is opposite to proposal in [4] where an event is encrypted using the lowest level node in the key hierarchy. In turn, a receiver node with a key at a higher level in the hierarchy must be able to derive the lower level key in order to be able to decrypt the received event. In such scheme, encryption and decryption are dependent on the consistency of the key indices

Overhead of Maintaining Consistent Key Indices

From another perspective, maintaining the consistency of the key indices results into two overhead costs. The first is the computations cost of updating the key indices in the key lattice itself. This cost is not eliminated for any of the previous design attempts, even in the volatility compatible version of the second design attempts, in section 4.6.4.

The second cost is the cost of unnecessary sending of key update messages to subscribers to nodes with no changes in the subscribers group. This issue is mainly regarding the second design attempt, as discussed in section 4.6.4. We proposed a solution for this issue in section 4.6.5, however the computations cost of updating the key indices was not eliminated and additional storage costs are required for the queue of mixed key indices.

Inconsistent Key Indices

To address the two above mentioned issues, we make use of the above mentioned notice of the independency of encryption/decryption of the consistency of the key indices. In turn, we propose the following design concept:

> ***Design Concept.9***
>
> ***An Inconsistent Key Index*** *at node n, in a forest instance, is defined as keyIndex(n), which is defined by Design Concept.1. It is inconsistent because:*
>
> $$keyIndex(p) \not\subset keyIndex(n)$$
>
> *, where p is the current parent of nod n in the forest instance, and according to the definition forest of filter coverage in forest, Defintinon.2 in this text, it's not possible that node n can use keyIndex(n) to derive keyIndex(p), which is a correctness property for the forest based key lattice.*

According to this design concept, in a forest based key lattice it is not necessary that the key index of node n is derived from the key index of its current parent node in the forest instance. Next, we define the operations the forest based key establishment.

Handling Connecting/Disconnecting Forest Nodes

In section 4.7, we reviewed the operations of addition and deletion of a node in the forest data structure. Also, we mentioned that the operations of connecting and disconnecting nodes, during addition and deletion of forest, are the most elementary operations which we build our forest based key establishment around.

Considering the above defined concept of *inconsistent key index*, we define the following procedure for creating a key index for a node in forest upon connecting it to a parent node:

Procedure.17

Let n be a newly added child node to node p in a forest instance. Then keyIndex(n) can be computed from keyIndex(p) as follows

If length(keyIndex(n)) = 0, then

a. Increment the numberOfChildren(p) by 1

b. Define keyIndex(n) = keyIndex(p) | numberOfChildren (p)

c. Compute encryptionKey(n) = ***Procedure.2(****keyIndex(n)****)***

, where length(s) returns the number of characters in string s, numberOfChildren(x) is the number of children nodes connected to node x in the poset instance.

In addition, we define a procedure for updating the key indices of nodes with only a changed subscribers group and not changed key index after the last *atomic update round*, which is defined by Design Concept.5.

Procedure.18

Let n be a node in a forest instance with a modified subscribers group and not a modified key index after a single atomic update round, then the process of ***Pre-Signalling Modification*** *for n is defined as follows:*

For every p = parentNode(n), call Procedure.17(p ,n)

, where parentNode(x) is parent node of node x in the forest instance and for Procdure.17 the first passed is the parent node in it and the second part is the child node in it.

Minimal Key Index Generation per Atomic Update Round

From the above two defined procedures, we notice that the generation of key indices is done only and only if it's necessary to generate a key index. In particular, when a node with no key index is added as a child of another node or when the subscribers group of a node changes without changing its key index after the last *atomic update round*. We define the procedure of signalling subscribers about the updates of key indices as follows:

> ***Procedure.19***
>
> *Let signallingSet be a set of to-be-signalled subscribers as defined in Design Concept.6, then the key updates signalling process for a subscriber s, where s ∈ signallingSet is defined as follows:*
>
> 1. *Compute signallingSet = signallingSet – {s}*
> 2. *Send keyIndex(subscribedNode(s)) and encryptionKey(subscribedNode(s)) to s, the sent message is encrypted by a shared secret with s*
>
> *, where subscribedNode(s) returns the node in the forest to which subscriber s is currently subscribing.*

To completely define our proposal of forest based key establishment, only a single functionality can be added to the above defined procedures, namely, efficient group key distribution using key hierarchies in the same manner as we used them in section 4.6.5, for minimizing the number of distribution encryption operations.

4.8.3 Usage

In section 4.6.3, we explained how the poset based key establishment proposal is used in a Pub/Sub infrastructure. The usage of forest based key establishment differs from the usage of the poset based one by just the used data structure which, i.e. forest in the first case and poset in the second one. No more explanation is required than of section 4.6.3.

4.8.4 Analysis

Finally, we analyse our proposal of forest based key establishment. We specify this analysis in the following points:

Inherited Advantages from Poset Based Key Establishment

In section 4.6.4, we analysed the poset based key establishment, our proposal in the second design attempt. From that analysis and from our review of poset, in section 4.4, and from our review of forest, in section 4.7, we conclude that proposal of forest based key establishment inherits a number of advantages from the poset based proposal.

Firstly, forest based key establishment inherits the generality of the key lattice to represent any dimension lattice, which represent the user specified privacy rules from the perspective of a given privacy dimension. Secondly, forest based key establishment inherits the consistency among the key lattice and the subscription lattice, which is forest in this case. In turn, no redundant encryptions are possible while encrypting for a secure multicasting. Finally, forest based key establishment inherits the fault tolerant clustered key management, which is achieved by trusting Pub/Sub brokers to act as key canters for their local subscribers.

Volatility Compatible Forward and Backward Security

Forest based key establishment goes beyond the inherited advantages form poset based key establishment by one more important advantage, namely, supporting volatility compatible forward and backward security. This feature is clearly defined by the *minimal key index generation per atomic update Round* in section 4.8.2. In turn, we conclude that forest based key establishment is a suitable proposal for a privacy aware key establishment in a ubiquitous Pub/Sub infrastructure.

4.9 Implementation and Results

In this section, we introduce the design attempts which we implemented prototypes for. Also, we implemented visualization tools to visualize the resulting key hierarchies/lattices. Finally, we discuss

4.9.1 Implementation and Data Structures

As proofs of concept, we implemented two prototypes. The first prototype is an implementation of the poset based key establishment which we introduced in section 4.6.2. In this implementation, we focus on implementing the mixed dotted key index, which we defined by Design Concept.4. Also, we implemented all the associated elementary operations and the nodes connecting and disconnecting operations in section. In figure.23, we illustrate the data structure for representing the mixed dotted key index {2.2.1, 3.1.1, 4.1.1} of a node which is the first child to both its parents {2.2} and {3.1, 4.1}.

The second prototype is an implementation of the forest based key establishment which we introduced in section 4.8.2. The implementation is simple because of the minimal key index generation feature of forest based key establishment. Also, the data structure for representing the dotted key index, which we defined by Design Concept.1, was simply implemented as an array of integers. The code of both prototypes is based on the implementation of poset and forest of the fuego core project [41]. All the implementation is this thesis is done in Java.

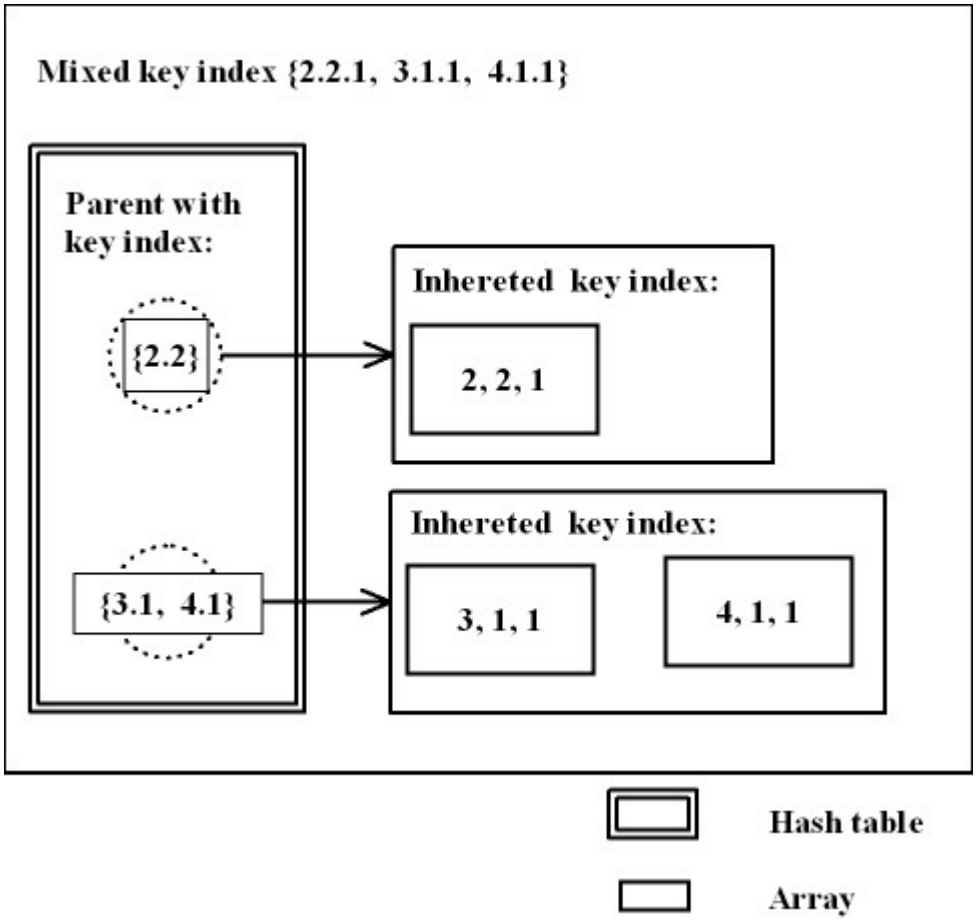

Fig.23. the data structure of a mixed dotted key index

4.9.2 Visualization of Key Hierarchies/Lattices

In addition we developed two visualization tools, one for visualising the resulting key lattices of the poset based key establishment and the other for visualising the resulting inconsistent key hierarchies from the forest based key establishment. In addition to the purpose of visualization, these two tools have been very useful for debugging the implementations, by verifying the visualized results against the expected ones. But in the case of poset based key establishment, formal correctness properties were defined to test its results. This is because of the complexity of this scheme makes testing it using the usual software debugging mechanism and the visualisation based debugging not sufficient.

In figure 24 a screen shot from the visualization tool of results of the forest based key establishment shows an example inconsistent key hierarchy under node with key index *90* which is part of the whole key forest. Horizontal and vertical scrolling can be used to reveal its other parts.

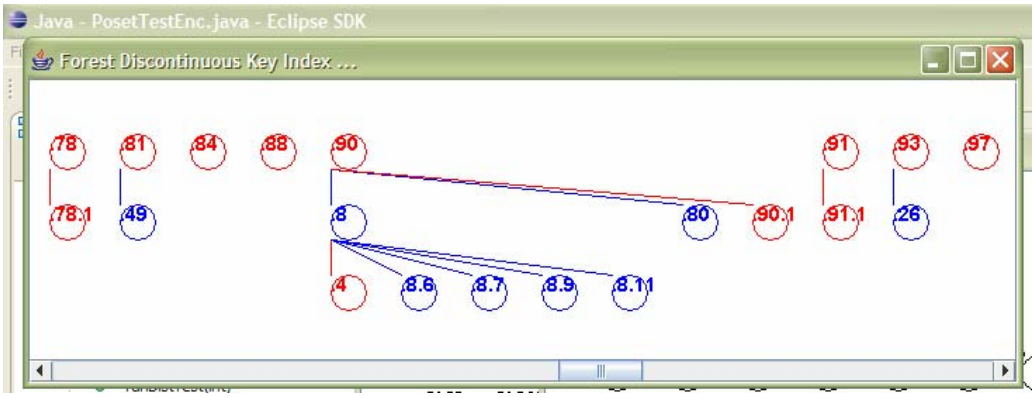

Fig.24. a result inconsistent key indices hierarchy from forest based key establishment

In figure 25, a screen shot of the visualization tool of the poset based key hierarchy shows the node with mixed key index *{156.1, 138.1, 144.3, 167.3.1, 163.4.1, 167.11.2}* and its parents along all the paths to the uncovered set of the poset. Oppositely, in figure 26, another screen shot of the same visualization tool shows the key hierarchy under the node with mixed key index *{138}* which is a parent to the target node in figure 25 in the previous screen shot.

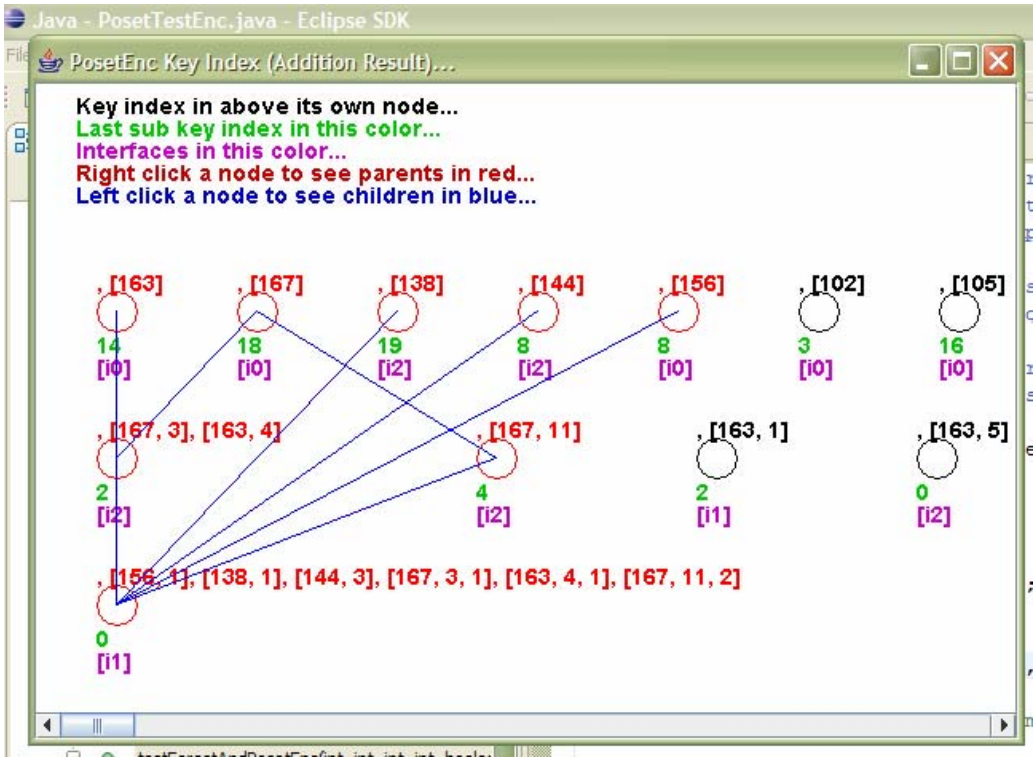

Fig.25. parent of node *{156.1, 138.1, 144.3, 167.3.1, 163.4.1, 167.11.2}*

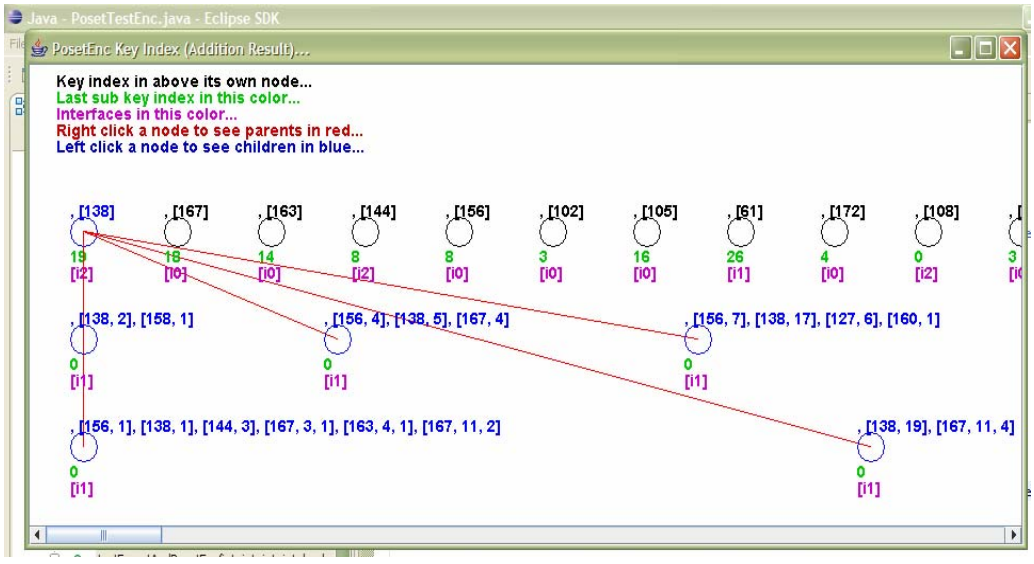

Fig.26. children of node *{138}*

Finally, in figure 27 three screen shots show the sequence of updating a simple mixed key indices lattice. The screen shout with label *1* shows the initial state, a trivial configuration of one parent one child totally ordered set.

Then, the screen shot with label *2* shows the change in the key lattice after deleting the second node, with subscriber *i10*, third node with subscriber *i20* replaces the removed node as the first child of the root node wit subscriber *i0*. As a result, this root node generates a new key index *{1.2}* for the newly added child, a change which is reflected to all subsequent nodes.

Finally, in screen shot with label 3, a new subscriber *i3* subscribes to the second node, which asks the parent for a new key index *{1.3}* to preserve backward security. Again, subsequent nodes are updated. This consistency in the key indices lattice in poset based key establishment is optimized in section 4.6.5 to minimize the number of required signalling messages to involved subscribers.

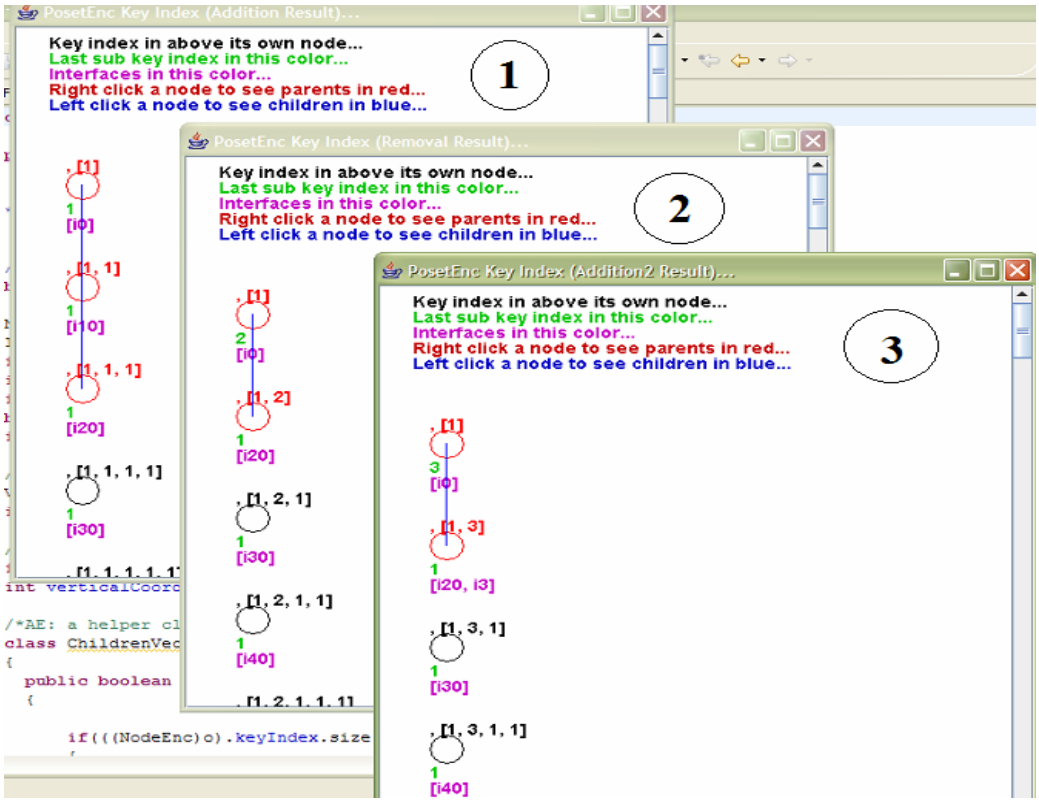

Fig.27. Consistent key indices lattices and forward/backward security

4.9.3 Correctness Testing of Poset Based Key Establishment

We define four correctness tests which must be passed by our implementation of the poset based key establishment in order to consider the resulting key hierarchies valid for any further type of testing. We list the correctness testing as follows:

a. Original Poset Consistency Test

This test compares the current status of a key generating poset instance, with added key establishment operations and associated key indices, to an original poset instance, without the additional key establishment operations or associated key indices. The comparison is done as on a node by node basis, where a node in the key generation poset instance is compared to a node in the original poset instance only and only if they are associated with the same subscription, i.e. the same set of filters.

To pass the test, the key generation poset instance must have the same number of uncovered nodes as well as the same total number of nodes as the original poset instance. In addition, each node in the key generation poset must have the same sets of predecessors and successors at those of the comparable node in the original poset instance.

b. Key Index Consistency Test

To pass this test, each node in the key generating poset instance must inherit every key element at every current parent node. Simply, the test checks whether every key element in every current parent node is a substring of a key element in the mixed key of the node under test. A key element in the mixed key index of the node under test is used for only once as an including string of a key element of a parent node.

c. Signal List Correctness Test

During the key indices maintenance operations, we continuously update a list of the nodes in the poset with updated key indices. This test checks the correctness of this list. To do so, the test records the values of the key indices at the start of a given atomic update round. Then the test records the values of the key indices at the end of the atomic update round. Finally, the test compares the key indices values at the start of the round to those at its end. As a result, it builds a list of the nodes with differences in their key indices. This list should be the same at the built list during the key indices maintenance.

The importance of this test comes from the reality that we modify existing poset implementation to add the key generation functionality. And we want to make sure that our implementation policy at lower level operations does not miss any of the key updates from the perspective of higher level operations, i.e. add and delete poset operations which are bounded by the atomic update round.

d. Interfaces Change List Correctness Test

It is a similar test to the previously defined one, i.e., signal list correctness test. The only difference is that it tests the correctness of a list of nodes with changed interfaces.

Chapter 5

Conclusion and Future Work

In this chapter, we conclude the thesis. In addition, we introduce a possible extension of the poset based key establishment, which we introduced in section 4.6.2. This extension may provide content based routing mechanism for encrypted events, which if proved to be practical it'll be can be a major contribution to the secure event routing in Pub/Sub systems.

5.1 Conclusion

In this thesis, we investigated the problem of generating encryption keys to secure event distribution to subscribers groups in Pub/Sub infrastructure for the ubiquitous environments. We focused on Pub/Sub because its interaction decoupling makes it a very significant candidate as a ubiquitous interaction paradigm. The volatile and error prone ubiquitous environment was studied in order to collect its requirements from the perspective of middleware infrastructures, e.g. Pub/Sub. In addition, we reviewed the literature for the current provided support by Pub/Sub to the requirements of ubiquitous infrastructures.

From all of these reviews, we concluded that Pub/Sub has been strengthened by proposals for defining and enforcing privacy policies in ubiquitous environments. Also, encrypting the disseminated events has been investigated for Pub/Sub by different approaches, which address the security requirements of the ubiquitous environments. However, the privacy and security underpinnings for Pub/Sub in ubiquitous environments were given a small intersection area, while they are much tied aspects to each other. So we had to investigate how to create encryption keys which reflect the privacy levels specified by the privacy rules in the ubiquitous infrastructures.

Consequently, the problem of generating encryption keys for subscribers groups in Pub/Sub infrastructures was investigated. Of course this is with the considerations of the dynamic and large group of subscribers in the ubiquitous environments. In turn, we concluded the problem definition as a group key establishment problem for dynamic and large groups, and several factors and

design consideration were investigated. Then we started our design attempts to solve the defined problem by addressing the concluded design considerations.

Afterwards, we started the first design attempt assuming that the key establishment is done by the same entity which maintains the privacy rules. By analysis, we concluded that we have to move the key establishment process to the local Pub/Sub brokers which receive subscriptions from clients. So, we started our second design attempt by mapping privacy dimensions at attributes in the poset instances incised the local Pub/Sub brokers. Finally, due to the complex operation in poset, which is expected introduce unsuitable overhead with respect to the volatility of the ubiquitous environments, we started our third and last design attempt using forest. The analysis of the forest based key establishment mechanisms shows a minimal number of operations, i.e. overhead, which is suitable for the purpose of this thesis. However, the poset based key establishment from the basis of our future work, which we discuss in the next section.

5.2 Future Work

In this section we introduce a possible extension of the poset based key establishment, which we proposed in section 4.6.2. The extension is targeted as a contribution to the secure event routing in Pub/Sub systems.
Firstly, we start by introducing our assumptions about handling user's privacy in entrusted Pub/Sub brokers' network. Secondly, we define our concept of secure event routing using mixed key indices.

5.2.1 Handling User Privacy in Entrusted Publish/Subscribe Brokers Network

In a ubiquitous environment, trust relationships between clients, e.g. subscriber mobile nodes, from one side and service providers, e.g. Pub/Sub brokers, from another side must be pre-established – by direct or indirect means [39]. In particular, mutual authentication between a client and a service should be established before accessing the service. Such authentication mainly requires at

least a piece of information about the user's identity. It my even require information about the user's location or other context information.

In case of entrusted service providers, e.g. entrusted Pub/Sub brokers, the access to such pieces of information may contradict with the user's privacy. At this point, traditional authentication mechanisms fail to work in the ubiquitous environment [39]. This is similar to the case for cellular networks [32], which we reviewed in section 3.4.6. In most cases, from the perspective of a service, it is only enough to know whether a client with a particular context is authorized for access or not, and not the exact information about the client.

Privacy preserving authentication has been proposed for the ubiquitous environments [39]. The proposed scheme employs *blinded signature*, which is a variant of the digital signature, and *hash chains*, which applies one-way hash computation, to protect the identity of a client from being revealed to a service during the authentication process. The scheme is an application layer one and it does not depend on any underlying security infrastructure.

Assuming the existence of such privacy-aware authentication scheme in a ubiquitous Pub/Sub infrastructure, we propose a possible extension to our poset based key establishment in the next section. We assume that this extension is a significant part of the contribution of this thesis.

5.2.2 Secure Event Routing in Entrusted Publish/Subscribe Brokers Network

In this section, we describe a possible scenario of secure content based routing in Pub/Sub infrastructures. We make use of the poset based key establishment scheme, the proposal of our second design attempt in section 4.5.

Scenario Entities

Firstly we define the interacting entities and their roles based on the proposal for privacy-aware security for cellular networks [32], which we reviewed in section 3.4.6**.** In particular, we map a User Entity, *UE*, to any Pub/Sub user

entity, i.e., publisher or subscriber. In addition, we map a Serving home Entity, *SE*, to any entrusted Pub/Sub broker. Finally, we map a Home Entity, *HE*, to a new node which we introduce to the Pub/Sub infrastructure, namely, *the Cluster Head Broker.*

A Cluster Head Broker, CHB, is a member of a trusted sub-network of Pub/Sub brokers. This sub-network is referred to as the *Trusted Core.* This sub-network forms a security management layer above other network in the same Pub/Sub network. Each *CHB* is responsible for maintaining the poset data structure as well as its derived key lattice for the subscribers at the set of entrusted Pub/Sub brokers which belong to its own cluster. A *CHB* is assumed to have a synchronized copy of the poset and the poset based key lattice of every other member of the trusted core. Clustering of brokers is assumed to be location based. In figure.28, we show a Pub/Sub network with the entities which we define in this section.

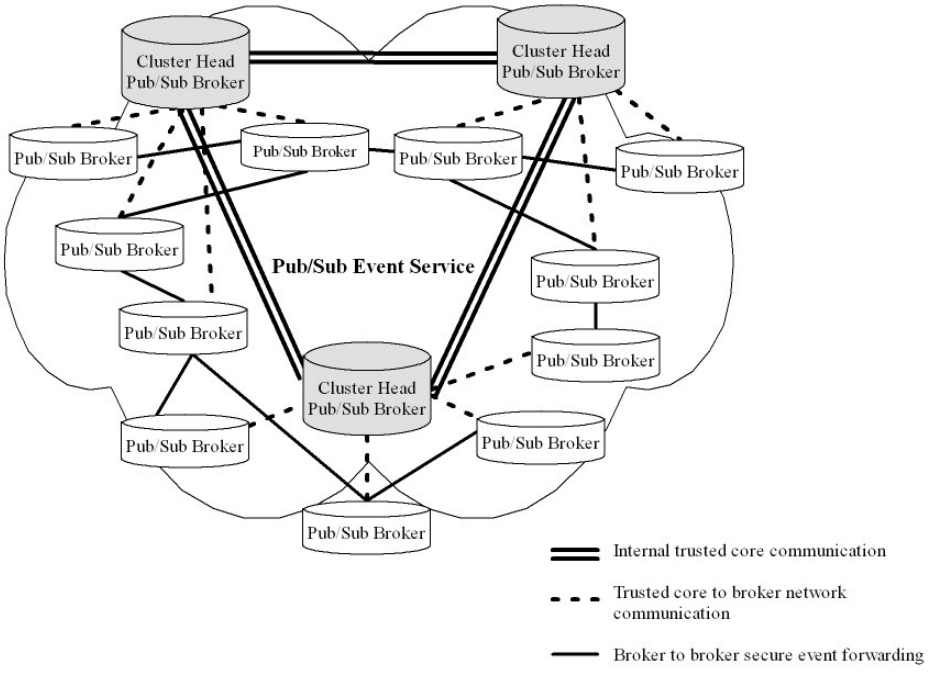

Fig.28. secure event routing using poset based mixed key indices

User Authentication at Entrusted Brokers

Secondly, we define the scenario of *secure event routing in entrusted Pub/Sub infrastructure*. A *UE* uses a privacy aware authentication process, similar to that we reviewed in section 5.2, to register to the nearest *SE*, i.e., the local entrusted Pub/Sub broker. The *SE* asks the *CHB* for access permission to the *UE*. After getting the access permission, there are two cases. In the first case, *UE* is an authorised subscriber, and in the second one, *UE* is an authorised publisher.

Handling Subscriptions at Entrusted Brokers

In the first case, *UE* sends an encrypted subscription to the *CHB* via the *SE*. The subscription is encrypted by a shared secret only by the *UE* and the *CHB*. In turn, the *CHB* updates its own subscription lattice and poset based key lattice to reflect the received subscription. Then, the *CHB* sends the mixed key index, which is associated with the node of the received subscription in it's the *CHB* poset instance, to both of the *SE* and the *UE*. It also sends the associated encryption key to the *UE*. All of these key update and key index update messages are confidentially protected in a way which ensures efficient group key distribution to the involved groups of *SE* nodes and *UE* nodes. Finally, the *CHB* synchronizes the copies of its poset and poset based key lattice at the other members of the trusted core.

The Projection of Cluster Heads to the Entrusted Brokers

Upon receiving the mixed key index from the *CHB*, the *SE* updates its own *projected lattice of mixed key indices*. This data structure contains subset of the mixed key indices in the key lattice at the *CHB*. It is formed by directly connecting each of these mixed key indices as a child to a parent mixed key index node. The first condition of selecting a parent node is to have a less number of elements in its mixed index than that of its child node. In addition, among all nodes which satisfy the first condition, the mixed key index of the selected parent node must have the largest number of common elements with key index of the child node.

After updating the relevant *lattice of mixed key indices*, the *SE*, as a Pub/Sub broker, forwards the received mixed key index to a neighbour Pub/Sub broker only and only if a covering mixed key index was not forwarded to it before, which is a typical poset forwarding policy. This way, at a given Pub/Sub broker, the mixed key indices of every *CHB* is projected by the *projected lattice of mixed key indices*.

At a Pub/Sub broker, the associated *projected lattice of mixed key indices* with a given *CHB* is uniquely identified from other projected lattices of mixed key indices at this Pub/Sub broker by *a unique CHB label*. This label is agreed by all members of the trusted core. The *unique CHB label* is also used inside each *CHB* to discriminate the synchronized copies of posets and poset based key lattices of different members of the trusted core.

The unique CHB label is used as a stamp in every *CHB* to *SE* message, every *SE* to Pub/Sub Broker message, and every Pub/Sub Broker to Pub/Sub Broker message, so that the receiver *SE* or Pub/Sub Broker can discriminate which *projected lattice of mixed key indices* it will operate on, e.g. for updating with a received key index or for matching against a received encrypted event.

Handling Publications at Entrusted Brokers

In the second case, i.e., an authorized publisher, the *UE* sends an encrypted event to the subscription to the *CHB* via the *SE*. The event is encrypted by a shared secret only by the *UE* and the *CHB*. Then, the *CHB* matches the event against all the poset instances it has, i.e. its own poset and the synchronized copies the posets of other members of the trusted core. This type of matching events is very similar to a proposal of matching events at the Pub/Sub network edge, i.e. early matching [40].

In turn, the *CHB* returns a set of encrypted copies of the event and their associated mixed key indices to the *SE*. The event is encrypted using the encryption key of every last matched node in every branch in every poset

instance. Every encrypted copy of the event is sent along with the *unique CHB label* of the poset base key lattice which was used for encryption.

Finally, the *SE* uses the returned mixed key indices and *unique CHB labels* to match the encrypted events against its own instances of projected lattices of mixed key indices. The matching is done by traversing a *projected lattice of mixed key indices* in a top-down approach, where a matched node is a node with any common key index members with the mixed key index of the being matched encrypted key. Upon forwarding the encrypted event, and its mixed key index and *unique CHB label*, to a subscriber Pub/Sub broker, this key index base matching operation is repeated. This process defines our future work of secure content based event routing.

References

[1] Weiser, M. 1995. The computer for the 21st century. In *Human-Computer interaction: Toward the Year 2000*, R. M. Baecker, J. Grudin, W. A. Buxton, and S. Greenberg, Eds. Morgan Kaufmann Publishers, San Francisco, CA, 933-940.

[2] Niemelä, E. and Latvakoski, J. 2004. Survey of requirements and solutions for ubiquitous software. In *Proceedings of the 3rd international Conference on Mobile and Ubiquitous Multimedia* (College Park, Maryland, October 27 - 29, 2004). MUM '04, vol. 83. ACM, New York, NY, 71-78. DOI= http://doi.acm.org/10.1145/1052380.1052391

[3] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. 2003. The many faces of publish/subscribe. *ACM Comput. Surv.* 35, 2 (Jun. 2003), 114-131. DOI= http://doi.acm.org/10.1145/857076.857078

[4] Srivatsa, M. and Liu, L. 2007. Secure Event Dissemination in Publish-Subscribe Networks. In *Proceedings of the 27th international Conference on Distributed Computing Systems* (June 25 - 27, 2007). ICDCS. IEEE Computer Society, Washington, DC, 22. DOI= http://dx.doi.org/10.1109/ICDCS.2007.136

[5] Richard Sharp, Kasim Rehman, "The 2005 UbiApp Workshop: What Makes Good Application-Led Research?," *IEEE Pervasive Computing* ,vol. 4, no. 3, pp. 80-82, July-September, 2005. DOI=http://doi.ieeecomputersociety.org/10.1109/MPRV.2005.69

[6] Issarny, V., Caporuscio, M., and Georgantas, N. 2007. A Perspective on the Future of Middleware-based Software Engineering. In *2007 Future of Software Engineering* (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 244-258. DOI= http://dx.doi.org/10.1109/FOSE.2007.2

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. *Interfaces and algorithms for a wide-area event notification service*. Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, Oct. 1999.

[8] Carzaniga, A. (1998). *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy.

[9] Kindberg, T. and Fox, A. 2002. System Software for Ubiquitous Computing. *IEEE Pervasive Computing* 1, 1 (Jan. 2002), 70-81. DOI= http://dx.doi.org/10.1109/MPRV.2002.993146

[10] Raatikainen, K., Christensen, H. B., and Nakajima, T. 2002. Application requirements for middleware for mobile and pervasive systems. *SIGMOBILE Mob. Comput. Commun. Rev.* 6, 4 (Oct. 2002), 16-24. DOI= http://doi.acm.org/10.1145/643550.643551

[11] Lukasz Opyrchal, Atul Prakash, and Amit Agrawal. Supporting Privacy Policies in a Publish-Subscribe Substrate for Pervasive Environments. . JOURNAL OF NETWORKS (JNW)
ISSN : 1796-2056 Volume : 2 Issue : 1 Date : February 2007 Academy Publisher, 2007

[12] S. Lederer, C. Beckmann, A. Dey, and J. Mankoff. *Managing Personal Information Disclosure in Ubiquitous Computing Environments*. University of California, Berkeley, Computer Science Division, Technical Report UCB-CSD-03-1257, July 2003.

[13] Borders, K., Zhao, X., and Prakash, A. 2005. CPOL: high-performance policy evaluation. In *Proceedings of the 12th ACM Conference on Computer and Communications Security* (Alexandria, VA, USA, November 07 - 11, 2005). CCS '05. ACM, New York, NY, 147-157. DOI= http://doi.acm.org/10.1145/1102120.1102142

[14] Pesonen, L. I., Eyers, D. M., and Bacon, J. 2007. Encryption-enforced access control in dynamic multi-domain publish/subscribe networks. In *Proceedings of the 2007 inaugural international Conference on Distributed Event-Based Systems* (Toronto, Ontario, Canada, June 20 - 22, 2007). DEBS '07, vol. 233. ACM, New York, NY, 104-115. DOI= http://doi.acm.org/10.1145/1266894.1266916

[15] Pesonen, L. I. and Bacon, J. 2005. Secure event types in content-based, multi-domain publish/subscribe systems. In *Proceedings of the 5th international Workshop on Software Engineering and Middleware* (Lisbon, Portugal, September 05 - 06, 2005). SEM '05. ACM, New York, NY, 98-105. DOI= http://doi.acm.org/10.1145/1108473.1108495

[16] Khurana, H. 2005. Scalable security and accounting services for content-based publish/subscribe systems. In *Proceedings of the 2005 ACM Symposium on Applied Computing* (Santa Fe, New Mexico, March 13 - 17, 2005). L. M.

Liebrock, Ed. SAC '05. ACM, New York, NY, 801-807. DOI= http://doi.acm.org/10.1145/1066677.1066862

[17] Jaeger, M. A. 2005. Self-organizing publish/subscribe. In *Proceedings of the 2nd international Doctoral Symposium on Middleware* (Grenoble, France, November 28 - December 02, 2005). DSM '05, vol. 114. ACM, New York, NY, 1-5. DOI= http://doi.acm.org/10.1145/1101140.1101144

[18] Belokosztolszki, A., Eyers, D. M., Pietzuch, P. R., Bacon, J., and Moody, K. 2003. Role-based access control for publish/subscribe middleware architectures. In *Proceedings of the 2nd international Workshop on Distributed Event-Based Systems* (San Diego, California, June 08 - 08, 2003). DEBS '03. ACM, New York, NY, 1-8. DOI= http://doi.acm.org/10.1145/966618.966622

[19] Wong, C. K., Gouda, M., and Lam, S. S. 2000. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.* 8, 1 (Feb. 2000), 16-30. DOI=http://dx.doi.org/10.1109/90.836475

[20] Opyrchal, L. and Prakash, A. 2001. Secure distribution of events in content-based publish subscribe systems. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10* (Washington, D.C., August 13 - 17, 2001). USENIX Security Symposium. USENIX Association, Berkeley, CA, 21-21.

[21] Xu, S. 2005. On the security of group communication schemes based on symmetric key cryptosystems. In *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks* (Alexandria, VA, USA, November 07 - 07, 2005). SASN '05. ACM, New York, NY, 22-31. DOI= http://doi.acm.org/10.1145/1102219.1102224

[22] Abdel-Hafez, A., Miri, A., and Orozco-Barbosa, L. 2006. Scalable and fault-tolerant key agreement protocol for dynamic groups. *Int. J. Netw. Manag.* 16, 3 (May. 2006), 185-201. DOI= http://dx.doi.org/10.1002/nem.592

[23] Lee, F. and Shieh, S. 2004. Scalable and lightweight key distribution for secure group communications. *Int. J. Netw. Manag.* 14, 3 (May. 2004), 167-176. DOI= http://dx.doi.org/10.1002/nem.515

[24] Li, J. H., Levy, R., Yu, M., and Bhattacharjee, B. 2006. A scalable key management and clustering scheme for ad hoc networks. In *Proceedings of the 1st international Conference on Scalable information Systems* (Hong Kong, May 30 - June 01, 2006). InfoScale '06, vol. 152. ACM, New York, NY, 28. DOI= http://doi.acm.org/10.1145/1146847.1146875

[25] Wang, W. and Bhargava, B. 2005. Key distribution and update for secure inter-group multicast communication. In *Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks* (Alexandria, VA, USA, November 07 - 07, 2005). SASN '05. ACM, New York, NY, 43-52. DOI= http://doi.acm.org/10.1145/1102219.1102227

[26] Di Pietro, R., Mancini, L. V., and Jajodia, S. 2002. Efficient and secure keys management for wireless mobile communications. In *Proceedings of the Second ACM international Workshop on Principles of Mobile Computing* (Toulouse, France, October 30 - 31, 2002). POMC '02. ACM, New York, NY, 66-73. DOI= http://doi.acm.org/10.1145/584490.584504

[27] Kuo, C., Studer, A., and Perrig, A. 2008. Mind your manners: socially appropriate wireless key establishment for groups. In *Proceedings of the First ACM Conference on Wireless Network Security* (Alexandria, VA, USA, March 31 - April 02, 2008). WiSec '08. ACM, New York, NY, 125-130. DOI= http://doi.acm.org/10.1145/1352533.1352553

[28] Li, D. and Sampalli, S. 2005. An efficient group key establishment in location-aided mobile ad hoc networks. In *Proceedings of the 2nd ACM international Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks* (Montreal, Quebec, Canada, October 10 - 13, 2005). PE-WASUN '05. ACM, New York, NY, 57-64. DOI= http://doi.acm.org/10.1145/1089803.1089967

[29] Du, W., Wang, R., and Ning, P. 2005. An efficient scheme for authenticating public keys in sensor networks. In *Proceedings of the 6th ACM international Symposium on Mobile Ad Hoc Networking and Computing* (Urbana-Champaign, IL, USA, May 25 - 27, 2005). MobiHoc '05. ACM, New York, NY, 58-67. DOI= http://doi.acm.org/10.1145/1062689.1062698

[30] Parnerkar, A., Guster, D., and Herath, J. 2003. Secret key distribution protocol using public key cryptography. *J. Comput. Small Coll.* 19, 1 (Oct. 2003), 182-193.

[31] Bresson, E. and Manulis, M. 2008. Securing group key exchange against strong corruptions. In *Proceedings of the 2008 ACM Symposium on information, Computer and Communications Security* (Tokyo, Japan, March 18 - 20, 2008). ASIACCS '08. ACM, New York, NY, 249-260. DOI= http://doi.acm.org/10.1145/1368310.1368347

[32] Køien, G. M. 2005. Privacy enhanced cellular access security. In *Proceedings of the 4th ACM Workshop on Wireless Security* (Cologne, Germany, September 02 - 02, 2005). WiSe '05. ACM, New York, NY, 57-66. DOI= http://doi.acm.org/10.1145/1080793.1080804

[33] Li Zhou Jinfeng Ni Ravishankar, C.V. *Short Paper: GKE: Efficient Group-based Key Establishment for Large Sensor Networks* In First International Conference on Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005.
Publication Date: 05-09 Sept. 2005 On page(s): 397- 399 ISBN: 0-7695-2369-2 Digital Object Identifier: 10.1109/SECURECOMM.2005.41 Date Published in Issue: 2006-03-20 10:17:39.0

[34] Jensen, C., Kligys, A., Pedersen, T., and Timko, I. 2004. Multidimensional data modeling for location-based services. *The VLDB Journal* 13, 1 (Jan. 2004), 1-21. DOI= http://dx.doi.org/10.1007/s00778-003-0091-3

[35] Sherman, A. T. and McGrew, D. A. 2003. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Trans. Softw. Eng.* 29, 5 (May. 2003), 444-458. DOI= http://dx.doi.org/10.1109/TSE.2003.1199073

[36] Rozeva, A. 2007. Dimensional hierarchies: implementation in data warehouse logical scheme design. In *Proceedings of the 2007 international Conference on Computer Systems and Technologies* (Bulgaria, June 14 - 15, 2007). B. Rachev, A. Smrikarov, and D. Dimov, Eds. CompSysTech '07, vol. 285. ACM, New York, NY, 1-6. DOI= http://doi.acm.org/10.1145/1330598.1330648

[37] Sasu Tarkoma, Jaakko Kangasharju. A Data Structure for Content-based Routing. In the proceedings of Internet and Multimedia Systems and Applications (EuroIMSA) 2005, February 21-23, 2005, Grindelwald Switzerland.

[38] S. Tarkoma, Efficient content-based routing, mobility-aware topologies, and temporal subspace matching, Ph.D. thesis, Department of Computer Science, University of Helsinki. 2006

[39] Ren, K. and Lou, W. 2007. Privacy-enhanced, attack-resilient access control in pervasive computing environments with optional context authentication capability. *Mob. Netw. Appl.* 12, 1 (Jan. 2007), 79-92. DOI= http://dx.doi.org/10.1007/s11036-006-0008-7

[40] Cao, F. Singh, J.P. *MEDYM: match-early and dynamic multicast for content-based publish-subscribe service networks,* in Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on Publication Date: 6-10 June 2005, On page(s): 370- 376, ISBN: 0-7695-2328-5 INSPEC Accession Number: 8598041, Digital Object Identifier: 10.1109/ICDCSW.2005.86, Date Published in Issue: 2005-06-20 11:37:49.0

[41] Sasu Tarkoma, Jaakko Kangasharju, Miika Komu, Mika Kousa, Tancred Lindholm, Marko Saaresto, Kristian Slavov, Thalainayar Balasubramanian Ramya, Kimmo Raatikainen. *Documentation of Fuego Service Set II Implementation,* Fuego Core Project, Helsinki Institute for Information Technology, February 17, 2006.